Aalto University
School of Engineering

Adopting Procedural Information Modeling in Urban Planning

Master's Thesis
Department of Real Estate, Planning and Geoinformatics,
School of Engineering, Aalto University

Espoo, 5th of May 2014

Bachelor of Science in Architecture Jussi Viinikka

Supervisor: Professor Kauko Viitanen
Instructor: D.Sc Kirsi Virrantaus

Aalto University
School of Engineering

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi

Abstract of master's thesis

Author	Jussi Viinikka
Title of thesis	Adopting Procedural Information Modeling in Urban Planning
Department	Department of Real Estate, Planning and Geoinformatics
Professorship	Real Estate Economics and Valuation
Code of professorship	Maa-20
Thesis supervisor	Professor Kauko Viitanen
Thesis advisor(s)	D.Sc. Kirsi Virrantaus
Date	05.05.2014
Number of pages	58
Language	English

Abstract

This thesis examines procedural modeling as a tool for urban plan creation. Procedural modeling historically has been used for 3D visualization of natural features, but with the release of the software CityEngine in 2008 the technology can easily be adopted also in problem domains dealing with urban environments.

The study begins with a requirement analysis conducted to explore the needs urban planning imposes on the technology, based on which a functional procedural modeling production system is built using the CityEngine platform and its Computer Generated Architecture (CGA) scripting language. A solution is presented to the problem of control in procedural generation methods by introducing the concept of a selectable “Level of Control” and how its implementation in the produced system enables the planner to flexibly assume the necessary amount of control over the generated model.

The finished product is then compared against the presented requirements of accuracy, efficiency, ease of use, high visual qualities, and advanced analytical capabilities. The efficiency of the system measured as the ratio between user interactions (mouse clicks and keystrokes) and modeling output in the setting of the assessment is found out to be two to three times greater than the efficiency of a more established manual modeling software.

The technology as demonstrated through the produced system is concluded to be especially suitable for preliminary land use studies estimating the building potentials of extensive land areas. Directions for future research with potential to expand the applicability of the technology are discussed.

Keywords CityEngine, Procedural Modeling, Urban Planning and Design Tools, Urban Models

Aalto-yliopisto
Insinööritieteiden
korkeakoulu

Aalto-yliopisto, PL 11000, 00076 AALTO
www.aalto.fi

Diplomityön tiivistelmä

Tekijä Jussi Viinikka		
Työn nimi Proseduraalisen tietomallintamisen käyttöönotto kaupunkisuunnittelussa		
Laitos Maankäyttötieteiden laitos		
Professuuri Kiinteistöoppi		**Professuurikoodi** Maa-20
Työn valvoja Professori Kauko Viitanen		
Työn ohjaaja(t) TkT Kirsi Virrantaus		
Päivämäärä 05.05.2014	**Sivumäärä** 58	**Kieli** Englanti

Tiivistelmä

Tässä diplomityössä tutkitaan proseduraalista mallintamista kaupunkisuunnittelun työvälineenä. Proseduraalista mallintamista on historiallisesti käytetty luonnonmuotojen 3D-visualisoimiseen, mutta vuonna 2008 julkaistu CityEngine-ohjelma mahdollistaa teknologian helpon käyttöönoton myös rakennettua ympäristöä koskevissa aihepiireissä.

Tutkielma alkaa analyysillä kaupunkisuunnittelun teknologiaan kohdistamista vaatimuksista, joiden perusteella rakennetaan CityEngineen ja sen Computer Generated Architecture (CGA) ohjelmointikieleen perustuva proseduraalinen mallinnusjärjestelmä. Ratkaisuna proseduraaliseen mallintamiseen liittyvään kontrollin problematiikkaan esitellään käsite valittavasta ”kontrollitasosta”, ja kuinka sen implementaatio toteutetussa järjestelmässä mahdollistaa suunnittelijan ottaa joustavasti tarpeellisen määrän kontrollia generoitavan mallin suhteen.

Valmista tuotetta verrataan esitettyihin tarkkuuden, tehokkuuden, käytön helppouden, korkealaatuisen visuaalisuuden, sekä kehittyneen analytiikan vaatimuksiin. Järjestelmän tehokkuus mitattuna käyttäjäinteraktioiden (hiiren klikkaukset ja näppäimistön painallukset) ja tuotetun mallin suhteena mittauksen asetelmassa on kahdesta kolmeen kertaa suurempi kuin vakiintuneemman manuaalisen mallinnusohjelman tehokkuus.

Proseduraalisen mallintamisen, sellaisena kuin se tuotetussa järjestelmässä on implementoitu, todetaan olevan erityisen sopiva alustavien rakentamisen määrää laajoille alueille haarukoivien maankäyttötarkastelujen tuottamiseen. Työn lopuksi käsitellään teknologian käyttöaluetta laajentavia tutkimussuuntia.

Avainsanat CityEngine, proseduraalinen mallintaminen, kaupunkisuunnittelun työkalut, kaupunkimalli

Preface

This thesis has been written as part of my work at WSP Finland in a Tekes funded research project BIMCity[1], aiming – among other goals – to *"establishing information model as a way of conduct in land use planning"*. At some point during the start of a new work project the question is always presented: with what program should we do this? The start of the master planning project of Sibbesborg in southern Sipoo made no difference. As I had kept an eye on the development of the program CityEngine I made a suggestion to my unit manager Teemu Holopainen that we should try applying CityEngine to the task. The risks of the undertaking stemming from CityEngine representing a rather unchartered technological territory would be limited as the funding would come from the BIMCity program to which the experiment would be a good fit. Also writing a thesis around the experiment would end my long search for a suitable topic. The suggestion was then approved and the work started.

I would like to thank Jan Wolski for initial discussions on the project and helping me get my hands on the student version of CityEngine, Matthias Buehler for answering my questions regarding CGA and other aspects of CityEngine on the software's discussion forum, professor Kirsi Virrantaus for commenting the thesis, and numerous people at WSP Finland's Lauttasaari office and elsewhere with whom I´ve had discussions in and around the topic of this thesis and questions of urban planning in general.

In Espoo, 5th of May 2014,
Jussi Viinikka

[1] Tekes, a publicly funded research and development financing agency is organized around various programs including public-private partnership based Strategic Centres for Science, Technology and Innovation (SHOK). RYM Oy, a SHOK for the built environment, organizes various research programs the first of which was the Built Environment Process Re-engineering (PRE) program implemented in 2010 – 2013. BIMCity then was one of the six work packages of the PRE program.

TABLE OF CONTENTS

List of Figures

List of Terms and Abbreviations

BIM	Building Information Modeling. A process for modeling a digital representation of a building using 3D-objects with attributes (e.g. a geometry defined as a wall of a certain material).
CAD	Computer Aided Design. The use of computers in the creation of a design.
CE	CityEngine. Procedural modeling software for urban environments.
CGA	Computer Generated Architecture. Scripting language used in CityEngine.
CRA	Constructive Research Approach. Methodological framework for research conducted to construct and evaluate an artifact.
DSR	Design Science Research. Methodological framework similar to CRA.
FAR	Floor Area Ratio. The ratio of building floor area to land area within some boundaries (e.g. a lot).
GIS	Geographic Information System. Computer system used to analyze geographical data.
GUI	Graphical User Interface. The component of software through which the user interacts with it using graphical elements such as buttons.
PM	Procedural Modeling. A set of techniques to create 3D-content algorithmically.
SDK	Software Development Kit. Set of tools to create applications based on a certain software framework.
UID	Unique Identifier. Identifier which is guaranteed to be unique (e.g. a serial number).

1 Introduction

1.1 Motivation and purpose of the thesis

The day-to-day operations in an urban planning consultancy to a large extent are inextricably intertwined with the digital tool case at the planner's disposal. Though there is a clear natural incentive to study the improvement of the tools the subject especially as it applies to creative tools rather than analytical tools seems to draw relatively little research and business interest. The reasons for this might include the limited size of the market[2] of which a large part is in the public sector without a clear financial incentive to improve the tools, the fragmented regulatory environment undermining the possibility for globally applicable products, the complex all-encompassing nature of urban planning itself, and perhaps the lack of IT-technology and computational methods in the educational curriculum of urban planning professionals. Still the work, be it design of a few blocks or a master plan for an extensive new area, needs to be done and today the work quite often is done with tools optimized for tasks other than urban planning and design.

The Sibbesborg project[3] is a prime example of this (Fig. 1). It started as an idea competition in 2011 for the planning of a new city of up to 100 000 people in southern Sipoo some 30 km from central Helsinki, and ensued as a statutory master planning project where WSP Finland was contracted as the consultant after winning the competition. The software used for planning of the area during the competition phase included at least PowerPoint (the main purpose of which is to create presentations), Microstation (a general purpose CAD-tool), AutoCAD (another CAD-tool), and ArchiCAD (a building information modeling software). None of the software is dedicated to aiding in the type of work urban planning and design is. Yet the same kind of questions kept coming up: How much could there be people in that area? Could the work intensive urban design part of the project be made any faster? Can we be sure the modeled cityscape roughly reflects the numbers of people proposed on the coarser zone based plans?

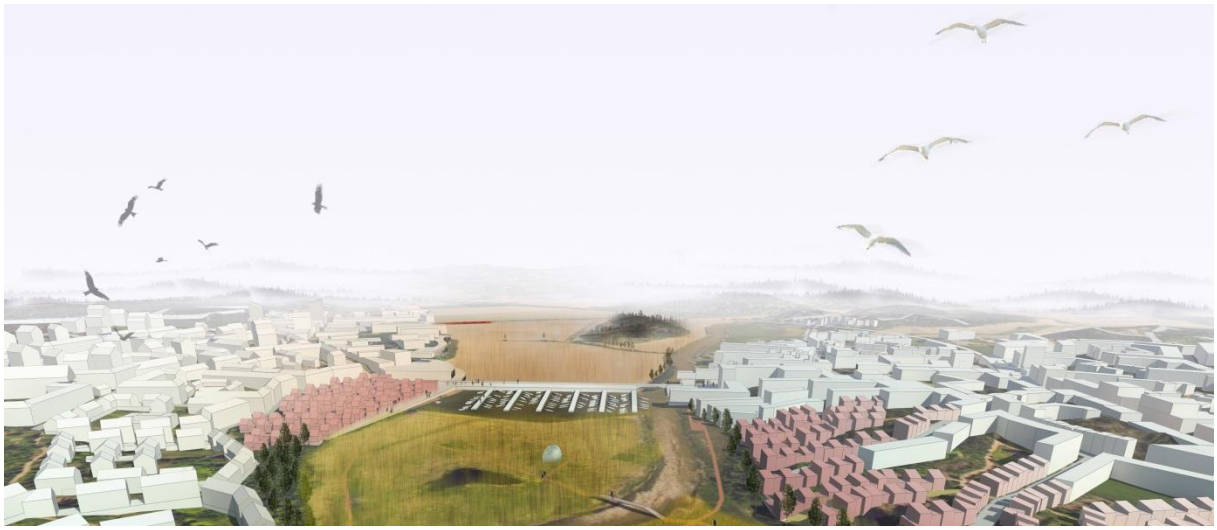

Figure 1 Sibbesborg project during the competition phase (WSP Finland, 2011)

[2] Recently the size of US market for software aimed for urban plan creation, visualization and preliminary analysis was estimated at $20 million annually (Synthicity, 2013). This compares with a global BIM market of $1.8 billion (Navigant Research, 2012).

[3] www.sibbesborg.net

Similar tasks are undertaken and questions presented during many planning commissions. Most of the questions are very basic and concern queries into the planned building volumes. Due to urban areas being aggregations of myriads of similar objects many of the tasks that an urban planning commission comprises of can be very repetitious in nature. In the course of planning various alternatives are presented, perhaps not officially but as possibilities the planner skims through. For example various drafts might be tried out to come up with the structure of some area of a plan. To understand the implications to cityscape, to better communicate the ideas, and for other such reasons the task might be undertaken in 3D. Despite the buildings of the area showing only minor differences in the numbers of floors, types of use, roof forms, or other such attributes, the modeling of each building would normally happen more or less manually without much efficiency derived from the fact that they all represent the object type "building". Similarly the calculations into the plans building volumes in the lack of better tools might have to be done by individually measuring building footprint areas, multiplying them with the number of floors, writing down the results in a spreadsheet, and hoping that the numbers won't get mixed up. Both of the operations would then have to be repeated for each variation of the plan. The fact that the needed operations are for the most part very simple and that there is a lot of them would seem to give support to the argument that the domain of urban planning could easily benefit from dedicated digital tools.

In fact, the market niche for urban plan creation software is not totally unexplored. Software such as Vianova's Novapoint Area Planning and Novapoint General Land Use Planning[4] though having their focus on the production of Finnish detailed and general plan documents respectively, include supportive features for both calculations and visualizations of the plans. Modelur[5], which seems to have been in a pre-beta phase for a number of years, is a more general purpose product built on SketchUp[6] enabling the parametric modeling of buildings and includes basic reporting functionality. Some products such as PixelActive's CityScape seem to have vanished from the market altogether[7]. CityCAD[8], software developed by a small company called Holistic City Software, aims to directly address the need for an integrated urban design and analysis tool, but its commercial success is unclear.

Contrary to the previous examples the strategy assumed by ESRI CityEngine[9] is to be both general enough to apply to the relatively large market of urban 3D-modeling (including domains such as the entertainment industry) but at the same time enable specialized use cases (e.g. master planning within some legislative framework) to find efficient workflows within the program. The strategy is based on introducing a modifiable middle component – a scripting language – in between the basic use of the program and the generated output. The scripting language allows the customization of the program to better answer to the specific needs of different use cases, while at the same time is simple enough to keep the script file development costs reasonable. In other words, rather than creating content based on user inputted parameters the program goes one step further by allowing the user to develop the procedures that take parameters. There is a trade-off however between the relative simplicity of the scripting language and the complexity of the procedures that can be created with it.

[4] http://www.vianova.fi/Tuotteet/Novapoint/Novapoint-Kaavasuunnittelu
[5] http://www.modelur.com/
[6] http://www.sketchup.com/
[7] http://www.prnewswire.com/news-releases/navteq-acquires-pixelactive-108659019.html
[8] http://www.holisticcity.co.uk/
[9] http://www.esri.com/software/cityengine

Procedural modeling, the modeling paradigm at the core of ESRI CityEngine, and its application to the domain of urban planning is the central topic of this study. The thesis describes the work done to adopt and assess procedural modeling as a tool to aid planners in the creation of urban plans. In the technological framework of CityEngine the task is only attainable through the creation of a script file that addresses the specific needs of the domain of urban planning. Though the initial motivation for the experiment was the Sibbesborg project within which the results of this thesis have been put to use, the produced script file is intended for general use. The research is then conducted by first studying the requirements the profession of urban planning imposes on the technology, constructing a procedural modeling production system on the CityEngine framework based on the requirements, evaluating how well the requirements were met by the produced system, and finally drawing the necessary conclusions.

1.2 Methodological framework

The type of research conducted in the thesis follows loosely the guidelines set by design science research (DSR) methodology. Vaishnavi and Kuechler (2013) give a comprehensive overview of the framework and the following definition:

"Design science research involves the creation of new knowledge through design of novel or innovative artifacts (things or processes that have or can have material existence) and analysis of the use and/or performance of such artifacts along with reflection and abstraction – to improve and understand the behavior of aspects of Information Systems."

Though the authors' focus is on Information Systems they make clear the methodology is applicable and has been applied – implicitly or explicitly and with varying degree of rigor – in numerous disciplines dealing with artificial constructions (such as architecture, engineering, management, etc.) as opposed to natural phenomena. The process model for a research project following the DSR framework is illustrated in Figure 2.

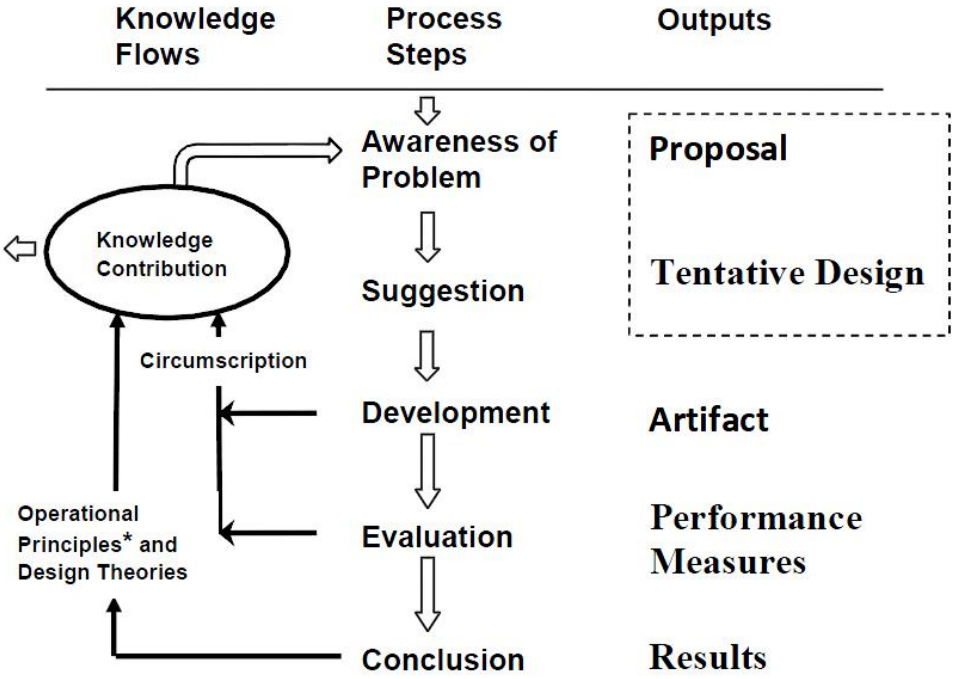

Figure 2 DSR process model (Vaishnavi & Kuechler, 2013)

In the thesis the problem from which the research starts was roughly outlined in the previous chapter as the digital creation of urban plans and designs. The finer details of the nature of the problem are explored in later chapters. The suggestion made was to apply procedural modeling to tackle the problem. Based on the suggestion an artifact is developed and eventually evaluated. The phases of suggestion, development, and evaluation in a standard process are then iterated giving rise to the alternative name of DSR as "Improvement Research" (ibid.). The results presented in the thesis however can only be understood as the first iteration of the development cycle, as the resources required for the continued improvement of the artifact are beyond the scope of this thesis. The evaluation results presented however give rather clear indications as to where the improvement efforts should be allocated on the next development cycle.

Improvement of a known solution however is not the only possible result of a DSR project. Gregor and Hevner (2013) present a more comprehensive framework for the types of knowledge contributions a DSR project can make as illustrated in Figure 3.

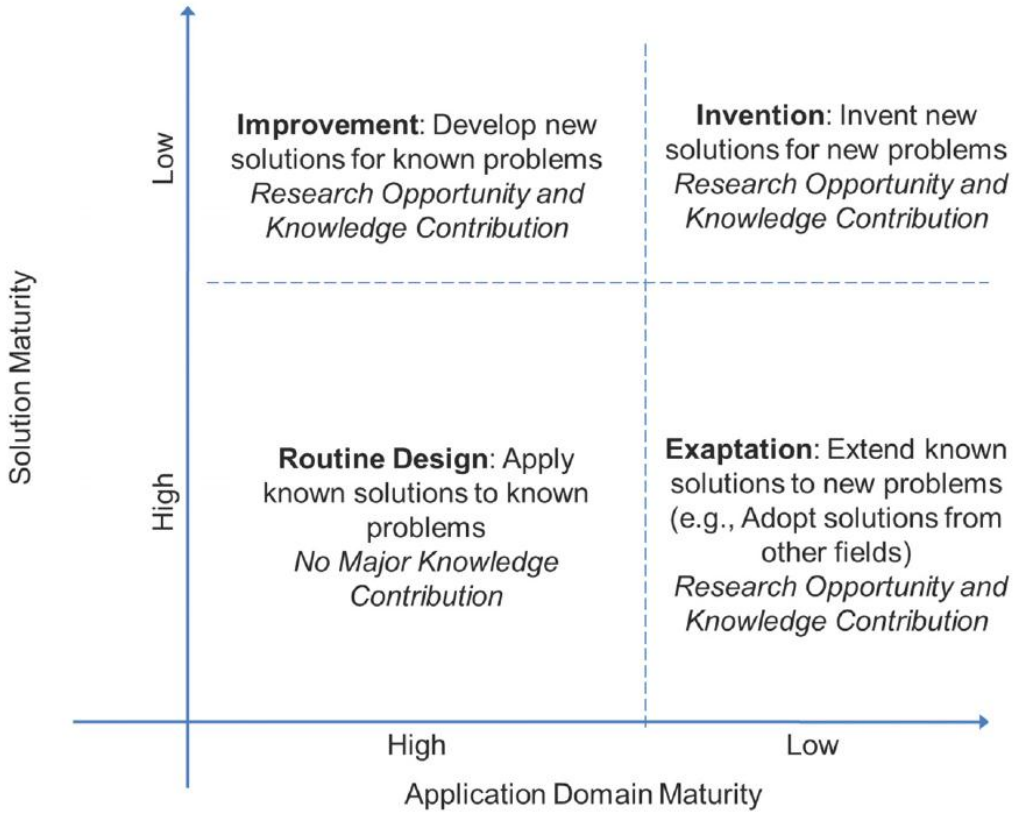

Figure 3 DSR Knowledge Contribution Framework (Gregor & Hevner, 2013)

In the context of the thesis the x-axis (application domain maturity) refers to the maturity of urban plan creation tools, and y-axis to the maturity of procedural modeling. Positioning the thesis on the matrix however is slightly problematic and depends on how the categories of "urban plan creation tools" and "procedural modeling" are understood. Urban planning, presumably with the aid of different tools, has been conducted since the ancient times. Time then could not have possibly limited the maturity of the toolset at a planner's disposal, and indeed some tools such as drawing equipment and scale models have stayed fairly constant throughout history. The digitalization of society however has in a way reset the development in urban planning and most other aspects of society, forcing the profession to adapt to the new

reality. The brief review conducted in the previous chapter on the dedicated urban plan creation tools on the market seem to indicate the maturity of *digital* urban plan creation tools to be somewhat low and leave plenty of room for innovation.

Procedural modeling (or procedural content generation generally) on the other hand, as shown later, is a paradigmatic term which since around the 1980s has found content for example in the domains of landscape and plant visualizations, and since the start of the new millennium also increasingly in domains dealing with urban environments (e.g. Smelik, et al., 2009). The maturity of the technology described by the term is then highly dependent on the exact domain where it has been applied. The software CityEngine is a relatively mature example of procedural modeling in the context of urban environments. However, as noted, it is a type of a semi-finished product requiring the user to create the script file that ultimately defines the content generating process. The maturity of CityEngine script files directly suitable for general urban planning purposes can safely be assumed to be somewhat low.

The aim of the thesis is to construct an integrated urban plan creation tool, in the sense that it would as well as possible facilitate both the consideration of initial data in the planning process and the creation of data reflecting the created plan for analysis purposes. The aim is intended to be achieved through the creation of a procedural production system using CityEngine and its scripting language. Perhaps more interesting than to speculate whether the aim represents a new problem or not, and whether the solution is already known or not (as differentiated in Figure 3), is to analyze the outputs of the research.

The artifacts produced during a DSR project are not limited to the final constructed and evaluated product, but importantly include the principles and constituent parts utilized in the product. March and Smith (1995) distinguish four different types of artifacts a DSR project can yield: constructs, models, methods, and instantiations. Gregor and Hevner (2013) complement the list with design theories, design principles, and technological rules, and group the artifacts to three different categories representing growing levels of maturity and abstractness of the produced knowledge. Instantiations, the working implemented products, form the lowest level of knowledge maturity in a DSR project. The produced script file and the system it is a part of is the instantiation of the proposed solution to the problem domain studied in this thesis. Though the production of a system to be applied to a specific project was indeed the primary aim of this thesis, the design of the system yielded other types of artifacts as well. The results presented and elaborated in chapter 5 introduce a number of these artifacts. The idea of a selectable "level of control" is a concept to tackle the issue of control in procedural production methods, and as a conceptualization of a problem within the domain of procedural modeling corresponds to the term construct in the artifact taxonomy. The internal organization of the modules which the script file consists of could be described as a design principle type of artifact. The algorithm which the script file uses to generate a building based on the user inputted FAR value and the lot the building sits on is a method type of an artifact. The constructs, design principles, and methods are examples of artifacts on the second level of knowledge maturity on the scale presented by Gregor and Hevner (2013).

Knowledge on the highest level of maturity, design theories, generally requires a sustained community effort to be formulated (Vaishnavi and Kuechler, 2013). Some traces of emerging theoretical contributions might be found in the chapter describing the requirements for an urban plan creation tool.

The concept of validity in DSR is closely linked to the utility of the produced artifacts. The validity of the results of the thesis is supported largely by observing the produced instantiation, and through controlled experimentation of its efficiency. The validity testing in a methodology related to DSR, the Constructive Research Approach (CRA) is based on market testing of the instantiations. According to Kasanen & Lukka (1993) there are three levels to the strength of the market test:

- **Weak market test:** Has any manager responsible for the financial results of his or her business unit been willing to apply the construction in question in his or her actual decision making?
- **Semi-strong market test:** Has the construction become widely adopted by companies?
- **Strong market test:** Have the business units applying the construction systematically produced better financial results than those which are not using it?

Though not a subject analyzed further in the thesis, it should be said that the produced system has passed the first market test and at the time of the writing is utilized in a number of projects at WSP.

1.3 Structure of the thesis

Chapter 1 highlighted the motivation, purpose, methodological framework, and the structure for the research conducted in the thesis.

Chapter 2 will present technical background information necessary to understand the following chapters. This includes information on procedural modeling in general, the type of procedural modeling implemented in CityEngine, and finally on the Computer Generated Architecture (CGA) scripting language used in the program.

Chapter 3 presents the research questions. The questions fall roughly into two categories: those concerning the design decisions to be made in the programming of the script file, and those analyzing the finished product.

Chapter 4 will focus on the methodology used to answer the questions presented in the previous chapter.

Chapter 5 will present the results of the research. The produced script file will be introduced and arguments made to support the design decisions made in the programming of it. After the results of the analysis on the expected requirements for the system are presented, the general structure of the produced script file is explained. An integral FAR-to-building algorithm is explained in detail, after which the results of the analysis on the performance of the produced system are presented.

Chapter 6 discusses the results on a requirement-by-requirement basis as analyzed in the previous chapter, and their implications to the stated goals of the study.

Chapter 7 concludes the thesis and points out some future directions in the study of the field.

2 Background information

2.1 Procedural Modeling (PM)

Procedural modeling is an umbrella term covering all the techniques to produce 3D content algorithmically. The procedural method however is not limited to 3D, but can be applied to the creation of digital content generally. The book "Texturing & Modeling – A Procedural Approach" by David S. Ebert et al. (2003) gives a thorough overview of the state of the field as it applies to digital graphics. Although procedural creation of audio content is also a studied subject, it is in the domain of digital graphics that the approach has been taken the furthest. It is also the domain most relevant for the topic of this thesis. To understand the nature and significance of the procedural approach in general, it is necessary to analyze what the theoretically unlimited number of techniques that it is comprised of share in common. David S. Ebert et al. identify the following features:

- *Abstraction*. Rather than explicitly creating and storing the end result of the modeling process, be it a geographical feature such as a mountain, or a human artifact such as a building, the procedural approach abstracts it into the underlying principles of the type. The abstraction, realized in the form of algorithms, can then be applied when and where needed.
- *Parametric control.* To control how the generalized abstraction is turned into individual 3D-models or other end results it is controlled through parameters. The number and type of parameters depends on the abstraction (i.e. the algorithm). In the case of a procedurally created mountain for example the height and ruggedness might be controllable parameters (Fig. 5), or number of floors and type of roof in the case of a building (Fig. 6).
- *Limited control.* As compared to manual content creation a substantial amount of controlling power is outsourced to the algorithms, the end user has a relatively limited amount of control over the end result. The lack of control has been proposed to be the main factor limiting the widespread adoption of procedural methods (Smelik, et al., 2009), and research has been conducted to tackle the issue (Smelik, et al., 2010).
- *Data amplification*. As content created procedurally can be stored as the sum of the algorithmic abstraction and the controlling parameters, rather than the end result itself, the amount of storage space required can be greatly reduced. How much space is saved depends on the complexity of the end result in relation to the complexity of the abstraction.
- *Efficiency.* The data amplification property can equally be understood as the ratio between work input and the resulting complexity of the output. Procedural modeling then generally is efficient.
- *Flexibility*. The algorithms generating content can be flexibly designed to meet any design requirements the end user might have on them. A procedural modeling algorithm can for example be anything from a physically accurate representation of a natural phenomenon, to a completely artistic endeavor.
- *Emergence*. Procedural modeling techniques exhibit emergence as their application can generate variety in a scale unreachable through manual content creation.

Procedural modeling can also be understood in comparison to manual modeling. In its purest form manual 3D-modeling might be described as the construction of polygonal mesh objects by creating their constituent elements one by one. In essence this would mean defining vertices (points in space), and connecting them with edges to create 3D-faces (closed sets of usually three edges). The 3D-faces would then define the surfaces of a 3D-object. Though this might have been a standard 3D-modeling workflow in the very early days of the technology, no current software operates solely on this basis. ArchiCAD, a building information modeling (BIM) software used in the architectural design of buildings, for example features a number of dedicated tools to create building elements. The wall tool can be used to model walls with their thickness and height given as numerical parameters and only the start and end points of the wall placed manually. The roof tool works by taking the roof thickness and angle as numerical parameters while the user defines the outlines of the roof element. Similarly in other software the tools to create 3D-content rely on techniques based on a varying level of automation. There is no clear cut distinction between procedural and manual modeling then, and the two might better be described as the opposite ends on a continuum of modeling techniques. In practical terms however a given software usually falls close to either end of the spectrum, with for example ArchiCAD being clearly manual and CityEngine being procedural.

2.1.1 The brief history of PM

Procedural generation techniques in digital graphics have been applied since at least the early 1980s. Due to the all-encompassing nature of the term, the history of the field is difficult if not impossible to summarize to a single narrative. Some generalizations about the development of the subject through the decades however can be made. The theoretical foundations for many of the techniques applied later were laid out in the 1960s and 1970s. This took place for example in the work of the Hungarian biologist Aristid Lindenmayer who developed a formal language called L-system to model plant growth (1968), the shape grammars developed by Stiny and Gips to generate art work (1971), and fractals as presented by Benoît Mandelbrot in his book “Fractals: Form, Chance and Dimension” (1977). All of these works obviously are based on previous research in mathematics and other fields, but as the development in computer technology at the time allowed the theories to transition fast to an increasing number of applications, they form natural milestones in the history of the field.

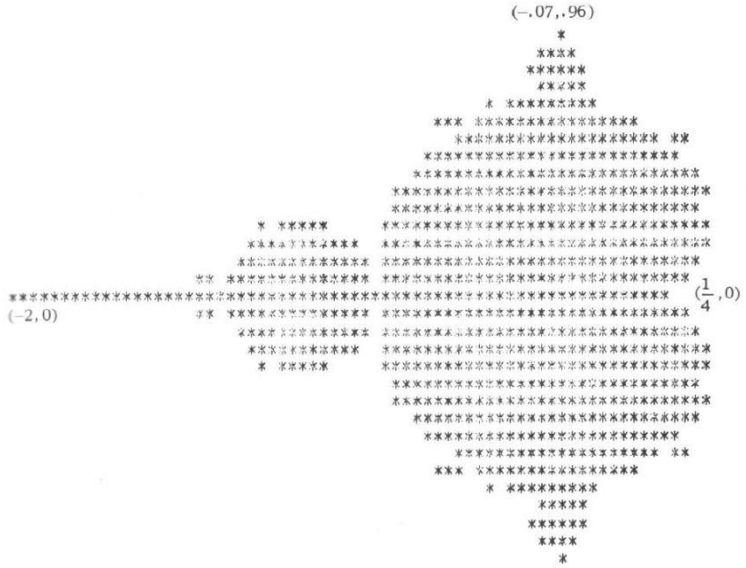

Figure 4 Mandelbrot set (Brooks & Matelski, 1981)

The first applications of the theories can be described as abstract: simple string rewriting sequences to illustrate L-systems and computer generated images of fractals such as the Mandelbrot set. The Mandelbrot set is a good example of the data amplification property. It is a collection of complex numbers defined by iterating the mathematical operation $z_{n+1} = z_n^2 +$ c on a complex plane where *c* is the complex number to be tested and $z_0 = 0$. If the magnitude of z_n remains bounded no matter how large *n* gets *c* is considered part of the Mandelbrot set. The set can then be visualized on a 2-dimensional plane by interpreting the real and the imaginary parts of *c* as the x-coordinate and the y-coordinate respectively and painting the coordinate points belonging to the set black (Fig. 4). The set, when zoomed into, yields infinitely complex self-similar but non-identical details *ad infitum* all compressed to the simple equation described.

After the abstract beginnings the 1980s saw the rise of numerous endeavors in digital graphics to generate imitations of different natural phenomena using techniques based on the mentioned[10] or other theoretical frameworks. Procedural generation of realistic textures for materials such as marble, wood, and stone were studied and topographic forms such as mountains were synthesized with procedural methods. The algorithms used for creating terrain elevation models are similar to the ones utilized in creating textures, such that they produce raster maps, using for example some type of a noise function. Instead of interpreting

Figure 5 Landscape created with fractal algorithms (Musgrave, 1989)

[10] It should be noted that modeling of plant growth was indeed the principal motivation for the development of Lindenmayer Systems in the first place

the values of the individual raster cells as color attributes however they are regarded as the cells height value, based on which a 3D-model of the terrain is created. The techniques to create these height maps are varied, but they all aim to produce results where the large scale topographic forms blend seamlessly into the finer details of the terrain profile (Fig. 5). This aim can be met for example by producing the height map as the sum of noise maps of increasing frequency and decreasing amplitude.

Eventually dynamic systems such as water, smoke, steam, and fire were also formalized to visually convincing digital representations. In recent years the most visible development in procedural methods has been in the generation of human made artifacts. Generation of such artifacts, for example brick wall textures, has of course been studied since the early days of the field. Only recently however have credible examples of more complex artifacts such as buildings or entire cities (Fig. 6) been presented.

Figure 6 A procedurally created city model for a commercial (Cliffhanger Visuals, 2010)

The domain of application for most of the techniques was - and still is – the entertainment industry. Especially the film industry has both applied and indeed created many procedural generation techniques as illustrated for example by the movie *Star Trek II: The Wrath of Khan* which already in 1982 showcases fire and landscapes created procedurally (Reeves, 1983). Another entertainment industry sector utilizing procedural generation techniques extensively is the game industry (see for example Hendrikx, et al., 2013). Whereas the film industry's motivation to use procedural generation has always been to create visually compelling effects, the motivation in the game industry initially was to minimize the storage space requirements of the games, as in the early days of personal computing memory resources were scarce. This led to procedural generation being applied to the most memory intensive elements of the games, i.e. the game worlds and levels. In practice this has meant generation of a variety of elements both graphical (such as world maps) and non-graphical (such as planet names and other text). As the memory constraints of computers were gradually eased and the graphical processing power increased, the focus in procedural generation in

games shifted more to individual graphical effects. Interestingly the same development in increasing memory and graphical processing resources however has recreated the demand to create game worlds procedurally as the increasing size and detail of the game worlds has translated into growing production costs of games (ibid.). The application of procedural content generation outside of the entertainment industry has been limited. It can be argued that as game-based technologies are increasingly utilized in other sectors of society, for example for military training purposes, the field of application for procedural techniques is indirectly widening. As direct applications procedural generation has enjoyed some success in architecture (for example in the works of Zaha Hadid), although mostly in the academia. The release of the program CityEngine in 2008 has sparked an interest to use procedural generation in a number of fields dealing with urban areas, such as urban planning and archeology (Dylla, et al., 2008).

2.1.2 Types of PM production systems

As Vanegas et al. point out (2010), the fundamental trade-off when selecting a procedural production system takes place between *expressiveness* and *efficiency*. The more freedom a producer requires to create variety in the content the stronger the expressive power of the production system needs to be. For example all of the application examples mentioned in the previous paragraphs could be produced using a programming language with a high level of expressiveness such as C. However, creating a procedurally modeled city starting from zero and using only C would require a lot of work and would not be efficient. To efficiently create the procedurally modeled city it would be better to use some domain specific – and thus efficient – application such as CityEngine for the task. On the other hand, using CityEngine would then limit the producer's ability to create other kinds of content, for example rendering creation of fractal terrains very difficult if not impossible.

Whereas CityEngine is a software and C is a programming language, what lies in between are the techniques, data structures, and generally the PM production systems used to create content procedurally. As discussed, terrains have been created procedurally using techniques based on fractals, usually by creating a height map with a fractal noise generator (Smelik, et al., 2009). Particle systems which define objects as clouds of primitive particles have found use in the modeling of dynamic elements such as fire, clouds and water (Reeves, 1983). The use of agent based models to simulate the growth of cities has been experimented, where the final product is the result of numerous agents interacting and following the behavior programmed into them (Lechner, et al., 2006). The optimization of urban designs with genetic algorithms to meet predefined design criteria is another field of experimentation (Wolski, 2010). As the technology in CityEngine is loosely based on L-systems, the focus in this study will however be on them.

2.1.3 L-systems

L-systems (or Lindenmayer systems) were developed by Aristid Lindenmayer as a way to model algae growth. An L-system is a parallel rewriting system and a formal grammar. It consists of a set of symbols defining the alphabet of the language, a collection of production rules defining how the symbols are rewritten to other symbols or strings of symbols, and an initial axiom string from which the production begins. The book "Algorithmic Beauty of Plants" (1990) by P. Prusinkiewicz, A. Lindenmayer et al. gives an illustrative and simple example. Given an L-system with the symbols A and B, the production rules A → AB

(meaning A is to be rewritten with AB) and B → A, and an initial axiom state of B, the first five iterations on the system yield the following results:

- Iteration 0: B
- Iteration 1: A
- Iteration 2: AB
- Iteration 3: ABA
- Iteration 4: ABAAB
- Iteration 5: ABAABABA

Due to the L-systems being *parallel* rewriting systems, all the applicable production rules are applied on each iteration. For example in the example given between iterations 4 and 5 the rule A → AB is applied three times and the rule B → A two times, simultaneously.

The alphabet of an L-system can also include constants, symbols for which there are no rewriting rules (except the implicit one where the symbol is rewritten with itself, e.g. A → A). Another example is an L-system with an alphabet consisting of variables 0 and 1, and constants [,], +, and -. With the production rules 1 → 11 and 0 → 1[+0]-0, and an axiom 0, the system during the first three iterations produces the following sequence:

- Iteration 0: 0
- Iteration 1: 1[+0]-0
- Iteration 2: 11[+1[+0]-0]1[+0]-0
- Iteration 3: 1111[+11[+1[+0]-0]-1[+0]-0]-11[+1[+0]-0]-1[+0]-0

The symbols of the alphabet can then be mapped to graphical operations for example using turtle graphics. Turtle graphics refers to a type of vector graphics where a cursor ("a turtle") moves and draws a line according to instructions given in relation to its own position and orientation. For the previous example the alphabet could be mapped to the following instructions:

- 0: draw a line segment by moving forward
- 1: draw a line segment by moving forward
- [: save position and angle
- +: turn right 45 degrees
-]: return to last saved position and angle
- -: turn left 45 degrees

Applying the instructions then to the axiom and the strings generated during iterations 1 to 4 would produce the following graphical representations presented in Figure 7.

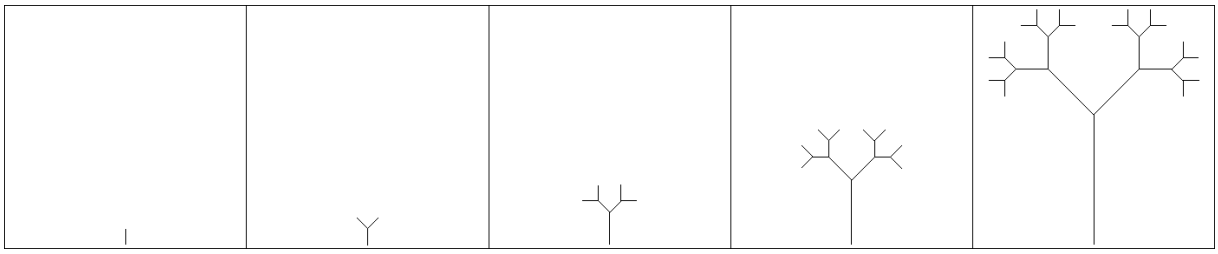

Figure 7 Turtle graphics applied to L-systems

Both of the examples represent the simplest type of L-systems, namely the deterministic and context free type. Being deterministic the production rule for a given symbol is always the same, and being context free the applied rule depends only on the symbol on which it is applied and not its neighbors. Since the L-systems were introduced however new variants have been developed.

Stochastic L-systems introduced probabilities to the selection of rules, so that a given symbol triggers a production rule randomly from a set of rules each with a predefined probability of being selected. An example stochastic production rule where A is rewritten B with a probability of 0.5, and C with a probability of 0.5 might look like the following:

$$A \rightarrow 0.5\ B$$
$$A \rightarrow 0.5\ C$$

In context-sensitive L-systems the production rule a certain symbol in a string triggers depends not only on the symbol itself but also on its predecessor (or predecessors) and successor (or successors). An example of a context-sensitive production rule where A is rewritten B only if the predecessor of the symbol A in the string is B might look like the following:

$$B < A \rightarrow B$$

In parametric L-systems the symbols have parameters that both influence the choice of the production rule and can be modified by the production rule. For example a symbol of the type A(x,y) triggers the following production rule if $x < 5$, which replaces the original symbol with A(x+1,y):

$$A(x,y) : x < 5 \rightarrow A(x+1,y)$$

Figure 8 Flower field created with L-systems (Prusinkiewicz, et al., 1990)

2.2 CityEngine (CE)

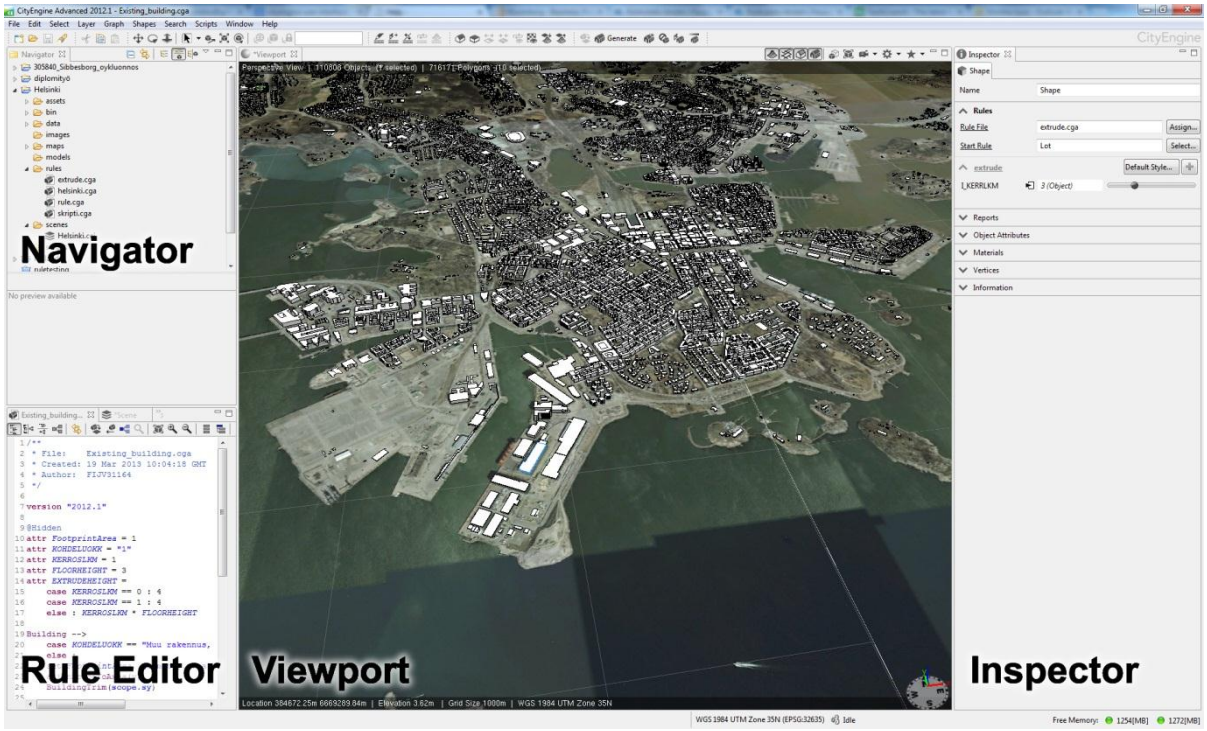

Figure 9 The CityEngine User Interface

CityEngine is a software application for the procedural generation of 3D city models. At the core of the software is a scripting language called Computer Generated Architecture (CGA), which is used to define the rules or procedures that create 3D content. CityEngine could be described as an integrated procedural modeling environment as it can be used for the whole procedural modeling development cycle. It includes tools from setting the scene up with for example pre-existing topographic data, to modeling street and lot geometries, to writing and analyzing the CGA script files, to applying the CGA files to create 3D models, and finally to exporting the derived model to desired post-processing programs.

2.2.1 History of CE

The development of CityEngine dates back to the master's thesis "Procedural Modeling of Cities" written by Pascal Müller in 2001 as part of his M.Sc. studies in computer science at the ETH Zürich (Müller, 2001). A presentation and a journal article soon followed at the 2001 SIGGRAPH conference (Paris & Müller, 2001). Müller then continued on the topic during his Ph.D. studies at the same university, and eventually a company called Procedural was formed around the CityEngine product. The company released the first commercial version of the software in 2008 with steady updates in the following years. In 2011 the company was acquired by ESRI, with the intention of integrating CityEngine with their ArcGIS geographic information system.

2.2.2 Type of PM in CE

CityEngine features three distinct PM systems:

- Automatic generation of street networks using a tool called "Street Growth Wizard"
- Algorithms to subdivide blocks into individual building lots

- CGA shape grammar to create 3D-content based on the created street and lot shapes

The automatic generation of street networks is based on an extended L-system as described by Paris & Müller (2001). The system differs from the discussed L-systems in a number of ways to account for the different topological nature of street networks as compared to the branching structures for which L-systems originally were developed. The street generation algorithm iterates a basic three step process to create the resultant network. At first the L-system produces an initial output of branching streets. Then an external *globalGoals* function is called to set the parameters of the generated streets according to the user inputted global goals regarding for example the type of street pattern aimed for (organic, grid, or radial). Finally the *localConstraints* function is called to finish up the iteration by deleting street segments in illegal areas (for example on water), creating intersections where two street segments cross, extending street segments to connect to other segments within a predefined distance, and by other such adjustments.

The subdivision of blocks to lots is done with one of the three algorithms: recursive, offset, or skeleton (Fig. 10). Depending on the algorithm they take parameters such as minimum lot width, irregularity factor, or maximum lot size. The recursive and offset subdivision algorithms create two types of lots, those with a street access and those in the middle of a block and without a street access. The skeleton subdivision algorithm however ensures the created lots always have one side connected to a street.

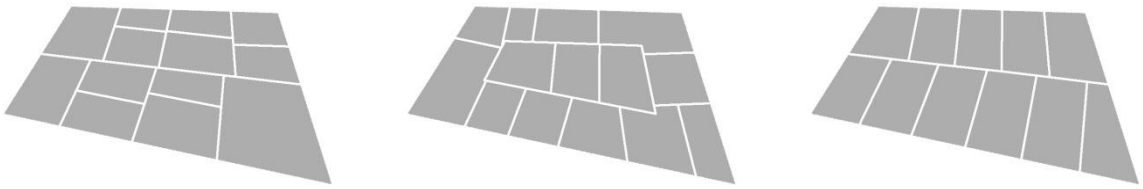

Figure 10 CE block subdivision algorithms: recursive, offset, and skeleton

The procedural production system used in CGA does not purely conform to any of the previously covered ones, but rather is an adapted and modified synthesis of aspects of at least L-systems (Lindenmayer, 1968), shape grammars (Stiny & Gips, 1971), and split grammars (Wonka, et al., 2003). Without simplifying too much, the production system of CGA could perhaps be described as an L-system which instead of symbols (such as A and B) operates on shapes, and instead of replacing shapes with other shapes (in the way that an L-system would rewrite A with B using the rule A → B) modifies and transforms them.

Similarly to L-systems the CGA production system also starts from an axiom, which in the usual CE workflow is a building lot. For example the axiom lot might trigger a production rule that splits the lot into the building footprint and yard element shapes. The building footprint shape might then trigger another production rule that extrudes the footprint shape to result in a 3D-building.

2.2.3 Basic structure of CE

From a usage point of view CityEngine is divided into roughly three parts. Firstly, it provides tools to create polygonal shapes that will later function as axioms for CGA in the procedural generation process. The creation of these shapes may happen through varying degree of control, ranging from fully manual to fully automatic. At the fully manual end of the spectrum

the polygons can be created by positioning their defining vertices one by one. More advanced modeling methods are based on creating graph networks, in essence interconnected lines. As CityEngine is not a general purpose procedural modeling application but one dedicated to creation of urban environments it comes with a set of built in features to aid in the task. Some of these features are based on interpreting the created graphs as street centerlines, which can then be given width as a parameter to create a polygon representing a street of the given width. If the graphs create closed loops CityEngine automatically forms a polygon representing a block in the closed area. Finally the blocks can be subdivided into individual lots by applying one of the discussed block subdivision algorithms. These modeling features are dynamic such that if a graph node (i.e. street end or intersection) is moved, the street, block, and lot polygons will be regenerated accordingly. At its most automatic CityEngine uses the mentioned "Street Growth Wizard" to generate street networks given a set of parameters, for example dictating the size of the street network to be created.

Secondly, CityEngine features tools to develop CGA script files. Though it is possible to write the script files with any plain text editor, the CGA rule editor integrated into the software provides features tailored for the CGA language. These include syntax highlighting and syntax error detection, and automatic code completion. Alternatively the rule files can be viewed and edited in a graphical node based mode.

Thirdly, CityEngine provides an environment where the written CGA script files (PM rules) can be assigned to the modeled initial shapes (PM axioms) to procedurally create the final 3D-model (PM data structure, i.e. a shape tree). If the CGA rules of a selected model include parametric control the parameters can be modified through the inspector window on the user interface (Fig. 9). The resulting model's data structure (shape tree, see Fig. 12) can be analyzed with a feature called Model Hierarchy Explorer.

To complement the three-step workflow described above CityEngine comes with a number of supporting features. These include the standard import and export functionality, which importantly supports geo-referenced GIS data such as ESRI Shape files. To automatize certain tasks and to overcome limitations of the CGA framework CityEngine also features a Python scripting interface.

Figure 11 Standard CE workflow

2.3 CGA-language

As noted, CGA is a language loosely based on an L-system data structure which instead of symbols operates on shapes to describe their transformations using operations written according to the CGA language specification. The CGA scripts are saved as plain text files with *.cga* extension. The initial shape from which the generative production begins is external

to the language and needs to be produced using the modeling capabilities of CityEngine or any other modeling software and later imported to CE. The shape transforming operations (extrusions, rotations, insertions of pre-modeled assets, etc.) are grouped into the production rules of the PM system, which function as flow of control structures. Depending on the design of the script, a CGA file might define numerous interlinked production rules. After a CGA file has been assigned to an initial shape the production rule from which the generative production begins needs to be defined. Both the name of the CGA file and the identifier of the first production rule to be applied – the start rule – are saved to the initial shape as attributes. The execution of a CGA script file assigned to an initial shape generates a shape tree data structure the leaves of which represent the resulting visible 3D-model (Fig. 12). These concepts are explained in more detail in the following chapters.

2.3.1 Initial shape

The initial (axiom) shape forms a necessary input to the procedural generation process, and the final 3D-model is a result of a sequence of transformative operations conducted on the initial shape. The initial shape has two primary attributes which connect it to the procedural generation process:

- Rule file: Defines the rule file assigned to the initial shape (e.g. script.cga)
- Start rule: Defines the production rule within the defined rule file from which the generative process begins (e.g. a production rule with the name Lot)

If the initial shape has other attributes (e.g. an imported building footprint shape with data attributes such as the number of floors) they can be used as parameters in the CGA production rules and operations as shown later.

2.3.2 Generated shapes

After the procedural generation process has been started (by assigning a CGA script file to an initial shape and clicking the generate button on the user interface of CE) the CGA data structure based on shapes is initiated. A shape is the basic building block of content generated with the CGA language. The first shape on the shape tree data structure – the root shape – is a geometrically identical copy of the initial shape. Geometry is however only one of the constituent elements of a shape. A shape generally consists of:

- Shape symbol
- Parameters
- Attributes:
    - Geometry
    - Oriented bounding box
    - Pivot
    - Other attributes

Shape symbol is the shape's name given to it in the production rule where it is created. In the case of the first shape however the name is given in the "Start rule" [11] attribute of the initial shape. It is not a unique identifier (UID), so numerous shapes with the same symbol and possibly differing sets of attribute values can exist simultaneously. If a production rule with a

[11] "Start rule" is a somewhat confusing name for the attribute, as although the attribute effectively determines the rule from where the generation begins, it would seem more consistent to call the attribute "Shape symbol" giving the first shape its name, which is then used to find a matching production rule.

matching name exists it is used to create the shape's successor shapes. If there is not such a production rule the shape will form a leaf on the shape tree data structure. For example if the name of the first shape in the "Start rule" attribute of the initial shape is defined as "Lot", the assigned CGA script file is searched for the production rule with the name "Lot". The "Lot" production rule might then generate separate shapes with the names "Footprint" and "Yard". In this case the script file would again be searched for production rules with matching names.

Shapes are created in production rules simply by giving them a name. For example a production rule which creates two shapes with the symbol B and one with the symbol C based on the predecessor shape A would look like the following:

$$A \rightarrow B\ B\ C$$

A shape however can also be created with *parameters*. CGA supports three types of parameters: Boolean, numeric, and strings. A shape's parameters are usually used to control the shape operations in the production rule which defines the shape's successors. For example a production rule which takes shape A as the starting point and creates two shapes with the symbol B giving them numerical parameters might look like the following:

$$A \rightarrow B(10)\ B(5)$$

Without a production rule which defines the shape B's successors the parameters would be meaningless. A following production rule might then be defined which takes shape B as the starting point and extrudes the shape by the amount defined in the parameter:

$$B(p) \rightarrow \text{extrude}(p)$$

As can be seen the actual parameters are given in the production rule which creates a shape, whereas the production rule which defines the shape's successors uses formal parameters. Also can be seen from the last presented production rule that it is not necessary to explicitly define the symbol of a successor leaf shape, but a production rule definition can end in a shape operation (e.g. extrusion). In this case the rule interpreter creates a leaf shape with the same symbol as its predecessor. The last presented production rule would then create a leaf shape B[12].

The shape *attributes* can be divided to those which the shape always and necessarily has, and additional attributes which the user can create. Of the first category are the shape's geometry, the geometry's oriented bounding box (i.e. scope), and the shape's pivot defining the shape's local coordinate system. The geometry is a mesh object, and includes the shader attributes such as color or texture used. The scope is defined relative to the pivot by the translation, rotation, and size vectors, each of which consists of sub-vectors for X, Y, and Z-coordinates. The pivot is defined by its position and orientation vectors (and their X, Y, and Z-coordinate sub-vectors). The necessary shape attributes then all relate to the shape's geometrical existence. The user can however create an arbitrary number of additional shape attributes of the types Boolean, numeric, or string by initiating them in the CGA script file. The following is an example of an attribute of the type string with the name "type" and value "modern" initiated in CGA:

attr type = "modern"

[12] This is also a bit confusing convention in CGA, as strictly speaking the B leaf shape should then result in an infinite loop

As the user created attributes are initiated per script file rather than per shape basis all the shapes created in a given CGA script file have all the attributes defined in the script file. If an attribute is initiated in the way presented it will be modifiable as a parameter in the inspector window of CE. If the attribute is intended only for the internal use of the production rules and shape operations in the script file the initiation of the attribute should be preceded with the annotation @Hidden, rendering the attribute invisible and unmodifiable to the end user. The values of the attributes can be changed with the set(*attribute, value*) shape operation within the CGA file. Changing the values of the attributes within the CGA code only affects the shape within which the changing is done and its successors in the shape tree data structure. All the shapes created in a script file then have all the attributes defined in the script file but their values might differ.

2.3.3 Production rules

The previous chapter illustrated various concepts related to shapes using simple production rules. The standard form of a CGA production rule (where PS = predecessor shape, SO = shape operations, and SS = successor shape) is as follows:

$$PS \rightarrow SO\ SS$$

The presented production rule will create a copy of predecessor shape, apply shape operations to the copied shape, and finally give it a new shape symbol (or name) sS. As discussed the shapes and production rules can however also include parametric control. The following is similar to the previous example, with the exception that the predecessor shape was created with parameters (p) which are then used to control the shape operations applied on the successor shape:

$$PS(p) \rightarrow SO(p)\ SS$$

The production rules can also include conditional statements. For example the successor shape to be generated might depend on some arbitrary condition (e.g. value of an attribute):

$$\begin{aligned} PS \rightarrow\ & \text{case condition}_1 : SS_1 \\ & \text{case condition}_2 : SS_2 \\ & \ldots \\ & \text{else} : SS_n \end{aligned}$$

Finally, the production rules can also include stochastic elements. A predecessor shape which is randomly succeeded by one of three successor shapes would take the following form:

$$\begin{aligned} PS \rightarrow\ & 33\% : SS_1 \\ & 33\% : SS_2 \\ & \text{else} : SS_3 \end{aligned}$$

The production rules defined in CGA can combine elements from all the covered production rule types. CGA does not directly support loop statements. Such a statement can however be constructed by combining parametric and conditional production rules in the following way:

$$\begin{aligned} PS(p) \rightarrow\ & \text{case } p > 0 : SO\ PS(p-1) \\ & \text{else} : SS \end{aligned}$$

2.3.4 Shape tree data structure

The created shapes are bound together by a data structure called shape tree. A shape tree defines the 3D-model generated from a single initial shape. Contrary to the way tree data structures are generally understood and used in IT the CGA shape tree cannot be rearranged, nor elements inserted to or deleted from arbitrary positions on the tree. The CGA shape tree is merely a hierarchical representation of the sequence in which the shapes of the model it depicts were created in the procedural generation process. The shape tree however can be visualized with a tool called Model Hierarchy Explorer. For example given an initial shape A and the following production rules the resulting shape tree would be as illustrated in Figure 12:

A → B
B → C C
C → D D E

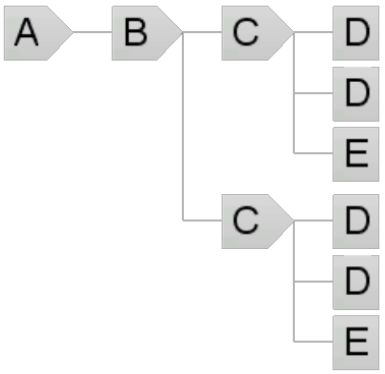

Figure 12 Shape tree data structure

The four D shapes and two E shapes are the shape tree's leaves. The leaf shapes represent the visible 3D-model. In other words only the geometry of the leaf shapes is visible in the CE viewport. Selecting an individual shape in the Model Hierarchy Explorer highlights the piece of CGA code (in the rule editor) of the production rule defining the shape's successor and selects the shape's geometry in the CE viewport window[13]. Although the inner nodes' (A, B, and C) geometry then is not normally visible, it together with its scope and pivot can be revealed by selecting the corresponding node in the Model Hierarchy Explorer. This is useful for analyzing and debugging the CGA code.

As discussed earlier all the shapes of a given model (i.e. all the shapes in a given shape tree) share the same set of attributes defined in the script file, but with possibly differing values. The script file of the previously given example could include the following attribute initiation:

attr type = "modern"

The value of the attribute could however be changed in the C → production rule in the following way:

C → D D set(type, "old") E

The set() operation would then change the value of the attribute "type" to "old" but only in the E shapes on the shape tree and their direct successors. Important aspect of CGA shapes that should then be understood is their ability to know of each other. It is very limited. In the given example D shapes have no way of knowing the value of the attribute "type" in the E shapes, and as E is not a UID with slightly different CGA code the shape tree could include E shapes each of which have a different value for the "type" attribute. This has implications as to the types of production systems that can be constructed with CGA.

[13] In essence, selecting the shape B in the Model Hierarchy Explorer highlights the result of "A → extrude(10) B" in the CE viewport, but "B → color(red)" in the CGA rule editor. This is another example of the confusion of whether it should be thought that shape B is defined in the previous or the latter piece of code.

The only structures in CGA that allow the shapes to know something of each other are the occlusion query functions which can be used to check whether a given shape's geometry is inside, overlaps, or touches the geometry of another shape. The another shape can be in the same shape tree as the checked shape (i.e. part of the model created from the same initial shape) or it can be a shape from another shape tree (i.e. part of a model created from a different initial shape).

2.3.5 Shape operations and other facilities

Whereas the production rules are flow of control structures, the actual shape processing is done using the shape operations of CGA language. The operations create new geometry, modify existing geometry, or operate on attributes and use functions without direct geometric results. The following paragraphs are a non-exhaustive list of these operations.

A CGA file can declare attributes, constants and mathematical functions that can then be used to control the geometric and other operations. The supported data types for attributes and constants are floats, strings, and Boolean values. The constants are only evaluated once during the procedural generation process and cannot be changed afterwards. The functions can be constructed using a selection of built-in functions, including for example basic mathematic, probabilistic, and geometrical query functions. The query functions can be used for example to query the volume of the current shape's geometry.

Both the modification of geometry and creation of new geometry take the existing geometry as the starting point. Geometry can be created for example by extruding the underlying polygon shape, or using one of the built-in functions to create roofs of different types. Geometry can also be created by inserting pre-modeled assets, in which operation the underlying shape only gives the position for the asset to be inserted.

The operations for modifying geometry either subdivide the shape's geometry to a number of new shapes on the shape tree, or manipulate the geometry without branching the resulting geometry to different shapes. CGA also provides functions for simple transformations such as moving, rotating, and scaling the shapes along with their geometries and scopes.

The operations can also specifically target the shape's scope, leaving the geometry in place. The scope modifying operations can then be thought of as a type of setup operations for the following geometry modifying operations. For example the scope could be modified to ensure that the extrusion direction is as intended, as the scope defines the shape's local coordinate system.

Texturing operations included in CGA provide tools for creating and manipulating the texturing coordinates for the generated geometries and projecting textures on the geometries using the created coordinates. CGA and CityEngine support six types of textures: color maps, bump maps, dirt maps, specular maps, opacity maps, and normal maps.

CGA also features a function for reporting arbitrary data (such as current shape's geometry's area) in real-time on the reporting window on the user interface (sub-window of the inspector window as seen on Fig. 9).

2.3.6 Parameter sources

The attributes, constants, and functions declared in the CGA files can be used to control the operations defined in the file. In other words they work as parameters to the operations. The rule files however are only one of four sources for the values of the parameters. Setting the parameter values through the inspector window in the CityEngine using slider buttons gives the user most control. The values can also be set using global attribute layers (i.e. bitmaps), for example defining certain areas of a CityEngine scene to be low-rise and some others high-rise. Finally, if the initial shapes contain attribute data (for example imported building footprints with the number of floors as an attribute), the data can be set as a parameter source controlling the CGA operations directly.

3 Research questions

3.1 Artifact and its analysis

The primary purpose of this thesis is to investigate the possibilities within and benefits of procedural modeling to urban planning. This question can only be answered within the limits of some implementation of procedural modeling for urban planning. The analyzed implementation then sets the lower boundary for the possible benefits. In other words the benefits procedural modeling might offer to urban planning are at least as great as in the analyzed implementation.

Software suitable for procedural modeling in urban planning is scarce. At the time of the writing to the knowledge of the author the only commercially available software application for this purpose is ESRI's CityEngine. As the software only provides a platform on which to use CGA files, CityEngine alone doesn't constitute an analyzable implementation of procedural modeling for urban planning. Although the standard installation of the software comes with a set of example projects along with their CGA script files, these projects and their script files mostly serve purposes other than general urban planning. Therefore, to answer the primary question of the thesis a secondary one is presented: how should a CGA script file for general purpose urban planning be designed?

These two questions are further specified as the following:

- What are the expected use cases and requirements the profession of urban planning imposes on an urban plan creation tool, and how should the requirements be addressed by creating a CGA script file within the framework of CityEngine?

- How well does the produced system meet the requirements, and particularly how efficient is it?

The question of efficiency of modeling with the script file is a specific examination of one limited aspect of the product. Other benefits such as the technology's potential to integrate the inherently multidisciplinary domain of urban planning, and it's potential to have an influence on the established urban planning processes are discussed.

3.2 Scope and limitations of the research

Urban planning is both universal and local. The focus in the thesis is on the technology of procedural modeling and aspects of urban planning which could be considered universal. It would not make sense however to refer to the technology's implications to urban planning processes without explicitly referring to a particular legislative planning system. The conclusions made in this thesis in relation to urban planning processes therefore should be understood in the context of the Finnish planning system.

The types of land use elements dealt with in urban planning include at least buildings and their lots, streets and other transport areas, and parks and other types of green areas. Although each of the different elements would merit their own specialized CGA script file, the focus in the thesis is solely on the buildings and their lots.

The requirements the use case analysis reveals are aimed at the whole system. In the technological framework assumed in the thesis the CityEngine platform however has to be

accepted as is, leaving only the CGA scripting component of the system modifiable. Also, the sole actor and the only intended user for the system is assumed to be an urban planning professional.

The software version used throughout this thesis is Esri CityEngine 2012.1. The behavior of CGA code might be different in other versions of the software.

4 Methodology

4.1 Design of the system

As the scope of the project from an IT development perspective is minimal – a single script file – following any formal software development methodology is not deemed necessary. However to ensure the general utility of the system the expected use cases and requirements towards the functionality of the system are surveyed.

The use case and requirement analysis for the system is conducted indirectly. Literature is reviewed to see how 3D-models and IT in general has been and could be used in urban planning, and what have been the perceived benefits and limitations thereof. The results of this analysis then form the base on which the script file is designed, with the intention of accommodating the revealed requirements as well as possible, strengthening the benefits and trying to overcome the limitations.

CGA is a marginal scripting language and as such there are no established ways to represent the structure of the script files written in it for documentation needs. It is also quite simple and after the introduction given in Chapter 2 presumably easily human readable. The approach assumed is then to mostly rely both on verbal descriptions and simplified CGA snippets to document the produced script file.

4.2 Analysis of the system

The focus of the analysis of the system in the thesis is on its efficiency. It is assumed that the time it takes to devise an urban plan draft is the sum of design thinking and using the design tools. Therefore to focus the analysis on the system – the design tool – the design thinking component is eliminated. This is done by introducing an existing benchmark area to be modeled using the produced system and for sake of comparison with a more established manual modeling tool.

Measuring the efficiency of a design tool as the ratio between time input and design (or modeling) output leaves a margin of error. The alertness of the designer, the processing power of the computer on which the measurement is conducted, unexpected events, and other such factors might distort the measurement. To overcome these sources of error the method applied in the thesis is to measure the efficiency as the ratio between user interactions and modeling output. The user interactions – mouse clicks and key strokes – are counted using software specialized for the task. To be relevant the efficiency measurements need to be balanced against the reached quality of the model. The evaluation of quality will be conducted visually.

Analysis on how well the other requirements are met by the produced system is conducted on a discussion level.

5 Results

5.1 Use cases and requirements

5.1.1 Legal context

The law governing urban planning in Finland, the Land-use and Building Act (Maankäyttö- ja rakennuslaki, 132/1999), includes a number of key elements relevant to the analysis of requirements on the designed system. These can be summarized as the following:

- *Planning hierarchy*. Urban planning is conducted on three main levels, where the regional plans guide the municipal master plans, which in turn guide detailed planning. Though the questions dealt with in each of the different levels have a degree of overlap the focus however is different, and thus sets differing requirements on the designed production system.
- *Public participation*. A major goal of the law is to ensure the public is both informed of plans under preparation and given a chance to participate in the planning processes. The implications to the requirements on the production system can be thought to relate for example to the understandability of the produced outputs.
- *Impact analysis*. To guarantee the plans are based on informed decisions the law requires the plans impacts to be analyzed from an array of perspectives. The requirement for the production system to accommodate analysis can be said to be deeply rooted in the planning law.

The planning levels most relevant for 3D-planning in the Finnish system are the local master plan devised for the whole or sub-area of a municipality to guide detailed planning, and the detailed plans usually the size of a few blocks guiding building permitting. The scale of regional plans is such that 3D-representations can be safely assumed to be irrelevant.

The use of 3D-modeling in Finnish master planning, be it for whichever use, underlines the requirement for the system to be efficient. The areas involved many times are so extensive, that the cost of 3D-modeling in most situations has been prohibitive. Partly for this reason 3D-modeling, despite being a rather standard method in detailed planning, has rarely been used in master planning. Instead, master planning has mostly relied on statutory zoning maps without clarifying illustrations. Often these have been followed by separate and more detailed non-statutory land use drafts that have then formed the baseline for the ensuing detailed planning. One use case for the produced system could be the production of these land use drafts. On the other hand, if the production tool is efficient enough, the need for these drafts might diminish as the contributions they provide could already be delivered in the master planning process. The role of 3D-modeling and other types of illustrations in master planning however is not clear. The guide "The Content and Representation of Master Plans"[14] published by the Ministry of Environment for example takes a critical stance toward illustrations of physical structures in master plans, arguing them to be too unreliable projections of the future (Salmi, 2006, p. 63).

In detailed planning the accuracy aimed for is an order of magnitude greater. At the same time, due to the much smaller areas involved, the efficiency of the 3D-modeling system is not

[14] Original title "Yleiskaavan sisältö ja esitystavat". Translation author's.

as critical. The accuracy is related to a number of factors, most importantly to the building volume represented by the 3D-model and mirroring the volume permitted in the plan, and factors having consequences on cityscape.

Public participation processes as relevant to 3D-modelling can be thought to benefit both from efficiency and accuracy. The latter criterion is a prerequisite for the understandability of the plans without which participation would be rather meaningless. The former makes it possible to have an influence on the 3D-plans as through an efficient production system they can be modified flexibly.

Impact analysis is largely dependent on the amount of information content in the plan and how fine-grained the information is. Information however is only the raw material for analysis. The analytical functions themselves could either be integrated into the production system or be based on the use of software external to the plan production system. In the latter case the requirement for the production system would be to facilitate the transfer of information to the external software.

5.1.2 General qualities

On a high level of abstraction there is only a single use case for the script file: to work as a tool for urban design and planning on different levels of detail. The level of detail mostly depends on the physical scale of the planning task, such that the larger the project the less detail is required. As discussed, 3D-planning itself is a detail relevant only in projects of a certain size range. The scale where 3D becomes irrelevant can be argued to be rather large however, as working with 3D-models covering vast metropolitan areas has been a common practice since at least the turn of the millennium (Batty, et al., 2000).

The study "Urban Planning as a Profession – Finnish Planners and the Challenges of the 21st Century"[15] by Puustinen (2004) provides valuable information on how the tools provided by IT have been used and perceived among Finnish planners. According to the study IT in general has had a positive effect on urban planning with only minor concerns. The main advantage of computer aided design the planners interviewed for the study identify is that of the produced plans being more accurate and reliable. The improvement in accuracy has led to the plans being more detailed, answering questions that before would have been solved only in latter planning phases. In other words the workload has concentrated in the beginning stages of a planning project, yet the production of alternatives is also perceived to be made easier by computers. All this, according to most of the interviewees has raised the quality of the plans.

Apart from the general concern the interviewed (ibid.) had about the time and effort it takes to learn the techniques IT makes possible, more specific problems were also mentioned: do the citizens understand the nature of urban planning visualizations as containing a significant amount of uncertainty, or are the visualizations and reality mixed up? Does the accuracy of the tools lead to too detailed plans, so that the legitimacy of the planning process is jeopardized with the apparent skipping of planning hierarchy levels? Do decisions depend too much on the quality of the illustrations?

[15] Original title "Yhdyskuntasuunnittelu ammattina – Suomalaiset kaavoittajat ja 2000-luvun haasteet". Translation author's.

5.1.3 Visual qualities

Regardless of the size of the 3D-planning project its main purpose is to give visual feedback on the qualities of it. The feedback can be continuous real time rendering (e.g. in the interface of a planning application), animations, or still images produced from the model.

The visualization capabilities of computers are also a major benefit according to the planners interviewed by Puustinen (2004). This advantage according to the interviewed was already evident in 2D environments, but the introduction of 3D-modeling has made the plans even greatly more illustrative. The planners interviewed point out that although the visualizations are especially important to the citizens and decision-makers they also improve the motivation of the planners themselves.

Visual representations of proposed plans, including 3D-models, are in key role in public participation processes where groups of people affected by a plan are engaged in the development of it. The relative merits of different digital visualization methods and their traditional counterparts have been studied by Al-Kodmany (2002). He concludes that public participation processes have the highest probability of succeeding when multiple different methods are used. According to Al-Kodmany the digital tools' ability to represent complex data flexibly at different scales and to easily document design alternatives produced during the public participation processes are a major advantage compared to more traditional methods. On the other hand working with pen and paper, physical scale models, and other such methods provides a level of interactivity and ease of access to laypeople not easily attainable by the relatively clumsy interfaces of digital tools (ibid.).

An important question regarding visualizations is the level of realism that should be aimed for. The findings in a study published in the Journal of Environmental Psychology on the representational validity of landscape visualizations (Daniel & Meitner, 2001) shed some light on the question. Although the research focuses on the study of scenic beauty of landscapes, the results seem applicable to aesthetic assessment of urban plans as well. The study begins by reminding of the purpose of visualizations: to anticipate human responses to the environments represented in the visualizations. The visualizations then as a tool are only valid if there is a strong positive correlation between the responses to the visualizations and the responses to the real environments represented in the visualizations. The researchers summarize a number of previous studies showing that there is indeed a strong correlation between responses to photographs of landscapes and responses caused by first hand experiences of the landscapes, with some important exceptions. In environments with strong dynamic elements such as rivers, or activity elements such as strenuous mountain hikes, the correlation is significantly weaker. The finding has important implications to the visualization of urban environments where especially traffic conditions form dynamic elements not captured by static illustrations. The researchers then conduct an experiment where a set of photographs of environmental settings where the aforementioned correlation based on the previous studies is strong is digitally manipulated to four categories representing different levels of abstraction. The categories range from the original photos to black and white sketches. The word "abstract" here should be understood in relative terms, as even the most abstract visualization – the black and white sketch – is clearly figurative. Groups of people are then asked to assess the scenic beauty of the landscapes illustrated in the visualizations. The study finds that the correlation between the assessments made using the original photos and the other visualizations is the weaker the more abstract the other visualization is, with no statistically significant correlation between the assessments made using the original photos

and those made using the black and white sketches. The conclusion the researchers draw is relevant also in the domain of urban planning: if the purpose of the visualizations is to communicate aesthetic qualities of the environment they represent, only visualizations with a high level of realism seem valid. Urban plans however also include qualities other than aesthetic that need to be visualized. These qualities, for example public transport accessibility, are however mostly and more naturally illustrated using maps rather than 3D-models.

5.1.4 Analytical qualities

Another high level use case for the script file is the analysis of the plans produced. As discussed, the type of analysis the production system would enable depends on the analysis tools the broader technology framework would provide and on the information content the script file would create to the model.

On all the statutory planning levels an important use case is probing and analyzing the land use potential of the planning area in terms of building volume. The accuracy aimed for in the assessment depends on the planning level such that the range of possible outcomes on the master planning level is deliberately wide, until it narrows down to an exact figure in the building permitting process. As the implications for traffic, economy, and other factors depend on the use of the planned building volume the different building types need to be distinguishable in the calculations.

The ease and accuracy with which the CAD tools enable the planners to calculate sizes of areas is another issue the planners interviewed by Puustinen (2004) brought forth. Interestingly juxtaposition was made between the accurate CAD measurements and those conducted with a mechanical tool called planimeter (Fig. 13). As especially master planning is based on zoning maps the calculations of areas often are not motivated by the need to know the size of the area per se, but the amount of potential building volume in the area (defined as the product of the size of the area and some FAR-value given to the area). Therefore a system that would enable the calculations to be conducted based directly on querying the floor areas of building geometries generated with the system would represent a further improvement in the accuracy of the calculations.

Figure 13 A Planimeter (Harvard University, n.d.)

The Land-use and Building Act sets certain requirements for the plans which the analytical functionality of the production system could help meet. These requirements as relevant to 3D-modeling on both the master and detailed planning levels are mostly related to the assessment of the plans impact on living conditions and environment, city- and landscape, built environment, and on health and security (Maankäyttö- ja rakennuslaki 132/1999). Different analysis studying these impacts where 3D-modeling could be of use include for example:

- *Cityscape and landscape analysis.* Studying a plans impact on physical cityscapes requires an idea on how the plan might materialize as buildings, which are best represented as 3D-models.

- *Shadow analysis.* The analysis whether or not a certain plan casts areas or buildings deemed critical in a physical shadow requires the plan to be modeled in 3D.

- *Viewshed analysis*. In a similar manner the visibility of a plan from surrounding areas or the views from particular points in the plan requires the plan to be modeled in 3D.

- *Acoustic analysis*. The effect a plan has on an area's noise conditions depends on the amount of noise it would generate (through traffic for example) and the configuration of physical structures, such as buildings, that might contain the noise. The configuration needs to be represented in 3D for reliable assessments.

Number of other analyses could be mentioned as well. The reporting of building volumes could be coupled with simple functions telling the amount of services required. The street network could be analyzed to estimate traffic conditions. Storm water management planning could benefit from information on the amount and distribution of pervious and non-pervious surfaces in the plan. Some of these analyses could relatively easily be conducted on CityEngine. Others should rather be handled in specialized software for the task.

5.1.5 Summary of requirements

Based on the previous exploration a number of requirements for the production system can be presented. The requirements can be divided to roughly three groups: general, visual, and analytical. Fulfilling these requirements would ensure that most of the probable use cases faced in urban planning could be adequately met. The requirements presented here could then be augmented with project specific requirements as needed and as made possible by the flexible nature of a CGA scripting based production system:

- *General.* The script file should be able to produce plans accurately, efficiently, and easily.

- *Visual.* The script file should be able to produce models with a high level of visual realism, but also include a mechanism to clarify the nature of the visualizations as containing a level of uncertainty. The requirement of visual capabilities could be further generalized as communicative capabilities, of which the visual is only one aspect.

- *Analytical.* The script file should be able to produce numeric data on the model, including building volumes by building type. For more complex analysis the interfaces and import/export workflows between the production system and the relevant analysis programs should be well defined.

5.2 General structure of the system

5.2.1 Level of Control

In the produced script file the critical architecture of control in the modeling process is handled by introducing the concept "Level of Control" (LOC). The LOC is an attribute in the CGA script file the value of which is set as a parameter by the user to determine the production rules the script file uses to produce the model. The script file has three LOC values in order of growing user control and decreasing efficiency:

- LOC 0, Indicators: The user defines the lot's floor area ratio (FAR), and the building's parameters such as number of floors and depth of the building are generated algorithmically.
- LOC 1, Parameters: The user defines the building's parameters directly, based on which the building geometry is generated.
- LOC 2, Custom: The user creates the building geometry manually.

The assumed approach introduces flexibility to the modeling process, such that the planner can rather freely move between efficiency and control. In planning projects of extensive areas this is important as it enables the model to be created efficiently while at the same time allows for critical subareas to be modeled with more design control. The LOC0 more or less corresponds to what Smelik, et al. (2010) proposed as the "declarative input approach" where the user assumes control over the generated model by setting high level constraints on the output, and what Vanegas, et al. (2012) call "inverse design" where the parameter space of the algorithms of the procedural model is automatically explored to find a set satisfying user inputted target indicator values.

In the current version of the script file the only implemented indicator is the floor-area-ratio (FAR), which the user can give as a parameter to a lot to create a building. Although it is a simple indicator its algorithmic implementation on the CGA language as seen later is already testing the limits of the language and its data structure. Implementing more complex and context dependent indicators then (e.g. regarding the visibility of some landmark) can be assumed to be out of the scope of CGA, and would require a component external to CGA and CityEngine to be realizable.

5.2.2 Level of Detail

The script file allows the shown output to be controlled with a "Level of Detail" (LOD) parameter. Similarly to the LOC parameter, the LOD value also controls the selection of production rules. The script file has three LOD values in order of growing richness of detail:

- LOD 0, Lot use: Lot areas distribution to green and paved surfaces.
- LOD 1, Massing: Buildings as plain volumes.
- LOD 2, Detailing: Fully detailed and textured buildings and landscaping features.

Although the requirement was presented for the produced model to be visually as realistic as possible, introducing less detailed alternative views to the model is justifiable. This is because some design questions, for example the general distribution of building volume over the planning area, might be more easily explored through a simplified view of the model.

5.2.3 Module structure

The LODs form modules of related production rules, so that all the landscaping is done in the "Lot use"-module, all the building massing in the "Massing"-module, and all the building detailing in the "Detailing"-module. Although every module consists of various production rules, each module has a single point of entry from the other modules. This design principle was taken to simplify the structure of the script file and to make the modules easily updatable.

The basic structure of a module begins with declarations of attributes visible to the user in the CityEngine user interface's inspector window (Fig. 9). The following lines then declare hidden attributes used by the production rules but not visible as parameters to the user. The rest of the module consists of production rule definitions. Apart from the general production rules responsible for the geometry creation, the modules finish by calling a module exit rule, which in turn calls the current module's freeze rule, reporting rule, and the following module's entry rule.

The freeze rule is used to create a visible copy (a leaf shape in the data structure) of the module's production before it is passed on for further manipulation in the following module. For example the lot's paved surface created in the LOD0-module is frozen as an independent copy but it is also passed on to the LOD1-module to form the basis for the creation of the building geometry as elaborated in chapter 5.3.

Although the reporting operations could be written dispersedly in the production rules, an approach is assumed to concentrate the module's reporting needs in a separate reporting rule functioning as a container. As the reporting rule also forms a leaf shape in the data structure (because no further production rules are called from it), its geometry is visible in the CE viewport. Showing the geometry of the reporting rule however is unnecessary and redundant, as it is already shown through the freeze rule. The redundancy can then be easily solved by explicitly deleting the geometry with a specialized CGA operation NIL which deletes the current shape from the shape tree.

Finally to move on to the next module its entry rule is called. A simplified version of a module of the defined type is presented below (Fig. 14).

```
visible_attribute =     default_value

@Hidden
hidden_attribute =      case LOC equals 0 : some_value
                        case LOC equals 1 : another_value
                        else : yet_another_value

LOD1_EntryRule →
    case LOD equals or is greater than 1 : LOD1_ProductionRule
    else : stop

LOD1_ProductionRule →
    case LOC equals 0 : some_operations(visible_attribute, hidden_attribute)
                        LOD1_ExitRule
    case LOC equals 1 : other_operations(visible_attribute, hidden_attribute)
                        LOD1_ExitRule
    else :  yet_other_operations(visible_attribute, hidden_attribute)
                        LOD1_ExitRule

LOD1_ExitRule →
            LOD1_FreezeRule
            LOD1_ReportingRule
            LOD2_EntryRule

LOD1_FreezeRule →
    stop

LOD1_ReportingRule →
    report(things relevant to the module)
    NIL
```

Figure 14 Module skeleton of the produced script file

The level of detail in the model then generally grows as the generative process progresses from the "Lot use"-module (LOD0), to the "Massing"-module (LOD1), and finally to the "Detailing"-module (LOD2). The structure is not without problems however. The texturing of the model for example is tied to the LOD-parameter such that the parameter values 0 and 1 produce models without textures and the parameter value 2 produces a model with textures. This is problematic as the lot use elements (e.g. the geometry of the paved surface) are not carried through to the LOD2-module where they could be textured, but instead as explained are "frozen" with the LOD1_Freeze rule. The dilemma could be solved either by decoupling the LOD-structure from the module structure and introducing LOD selection within the modules, or breaking the "single entry rule" principle.

The choice on the functional use of the generated building is done in the LOD0-module (as it is more an attribute of the lot than the building itself) using the categories "residential", "retail", "office", and "industrial". The choice is done separately for the ground floor and the upper floors to allow for mixed use buildings where for example the ground floor is retail space while the upper floors are residential. The choice has two effects: on the numbers on which the reporting operations are based (for example on the amount of floor space per one car parking place), and on the textures used for the building in the LOD2-module.

5.2.4 Reporting

The reporting is done following two principles: what the produced building and the lot accommodates, and what they demand. The reporting then forms a type of a balance sheet for the plan. For example the amount of paved surface on the lot determines the number of parking spaces the lot accommodates, and the size and type of the building determine the amount of parking spaces the building demands. Most of the reported figures are in some proportion to the calculated floor area of the generated building. For example by default the number of people a residential building has the capacity to accommodate is calculated by dividing the floor area by 50 m^2, and the amount of retail space demanded in the plan is arrived at by subsequently multiplying the number of people by 2.7 m^2 [16]. The default values are implemented as visible attributes, so they can be changed by the user in the CityEngine user interface.

5.2.5 Interfaces with external software

For further analysis in 3rd party programs, the buildings should be exportable with the reported attributes. From the export file formats available in CityEngine ESRI Shapefiles are most suitable for the purpose. The shapefile export function in CityEngine however only exports the underlying initial shapes of the models, i.e. the lot polygons. The data in the exported shapefiles is limited to the attributes the initial shapes had originally, and thus any data – such as the report data – created during the procedural generation process is not transmitted in the export.

The problem can be bypassed by recreating the reported data as attributes to the initial shapes. Initial shape attributes can be created manually by selecting the initial shape and clicking "Add Object Attribute" in the CityEngine inspector window. The process however can be automated with a script using the Python Scripting Interface. In the produced system a Python script is utilized which copies the report data as attributes to the initial shapes. ESRI Shapefiles representing building lots and including information on for example the amount of building volume planned on the lot can then be exported and further analyzed in any software supporting the format.

For some analysis purposes polygons representing building footprints are favorable over lot polygons. To enable the building footprints to be exported the produced CGA script file includes a Boolean parameter *ExportFootprints*. If set "true" the script file at end of the generative process deletes rest of the geometry leaving only the bottom faces of the buildings (i.e. the footprints) in place. To convert the footprint shapes to initial shapes (and thus make them exportable as ESRI Shapefiles) the user must still select the footprints and choose the

[16] 50 m^2 is a fairly standard figure used in Finnish planning. The amount of retail space was calculated using figures given by Statistics Finland accessible here:
http://tilastokeskus.fi/til/kamv/2009/kamv_2009_2011-04-29_tie_001_fi.html

option “Convert Models to Shapes” from the CE user interface. In the process the report data once again is lost as the created shapes don’t have any attributes. Programming a Python script which creates footprint shapes with the reported data as attributes might be possible.

5.3 FAR-to-building algorithm

5.3.1 Purpose of the algorithm

Most of the production rules, especially on LOC 1 and 2, perform straightforward tasks using the built-in functions of the CGA language. On LOC 0 (the indicator based level of control) more algorithmic approaches are however necessary. The FAR-to-building algorithm creates buildings based on user inputted FAR-value and a lot geometry. As there are numerous ways how a building satisfying the FAR value could be built on a lot an algorithm is needed to narrow down the possibilities. The purpose of the algorithm is to create buildings efficiently yet accurately reflecting a given target volume, based on minimal number of user inputted parameters. The requirements on the FAR-to-building algorithm are thus:

- to create a building the size of which accurately reflects the building volume dictated by the FAR value and the size of the lot it sits on
- to ensure the created building is credible in terms of its proportions
- to give necessary amount of control over the end result to the user.

Fulfilling only the first requirement is not enough as arbitrarily large building volumes could be created by giving a tiny building footprint a large number of floors. The second requirement to the algorithm then ensures that the building’s depth, height, and width are in realistic proportions to each other. As a building of the correct size and credible proportions still has a number of ways how it could be sited on the lot, the third requirement relates to giving the user control over the building siting.

5.3.2 Inputs to the algorithm

The algorithm needs four inputs to generate a model:

- FAR-value given as a numerical parameter
- polygon representing a building lot
- polygon representing the area where the building is to be sited
- parameter indicating how the building is to be sited on the given polygon.

The FAR-value is a simple float value (e.g. 0.5). The parameter is given through the inspector window on the CityEngine user interface (Fig. 9). The lot polygon is an initial shape created with the modeling tools of CityEngine. The second polygon input could be the initial lot shape created with CityEngine, but in the produced script file it is the part of the lot polygon which in the previous steps of the generative process has been designated as the *building area*. The building area is the part of the lot where in regulative terms building is permissible. Geometrically it is defined as the part of the lot polygon which is further than a given distance from the borders of other lots. The distance is set in a parameter *offset* which by default is 4 meters in accordance with typical Finnish detailed plans. In CGA terms the building area shape triggers the first production rule of the FAR-to-building algorithm.

The building siting parameter takes the value “both fronts”, “front”, “back”, “left”, or “right”. The parameter indicates the border of the building area as seen from the street along which the length dimension of the building aligns. A building aligning to the street would then take

the parameter value “front”, a building in a 90 degree angle to the street would take either “left” or “right”, and building at the back of the building area would take the value “back”. If a building in the corner of two streets is wanted to align to both streets (forming an L-shaped building) the value “both fronts” should be used.

5.3.3 Sequential steps

To generate buildings based on the inputs the following steps are taken:

1. *Target floor area* is calculated by querying the area of the lot polygon and multiplying it with the FAR value. If the target floor area is less than a predefined minimum the algorithm stops.
2. *Depth* for the building mass is calculated based on the target floor area. For target floor areas less than a predefined minimum a predefined minimum building depth is returned. For target floor areas greater than a predefined maximum a predefined maximum building depth is returned. The values in between are calculated using a linear interpolation function.
3. *Single floor area* for the building is calculated by first creating a footprint polygon defined by the depth value, the building siting parameter, and the building area polygon. The resulting footprint polygon’s area is then queried.
4. *Basement floor area* is calculated by generating the basement geometry and dividing the queried geometric volume with a predefined floor-to-floor height. The basement is thought to have a uniform base floor at the level of the footprint’s lowest point. The basement floor area is thought to be formed of the part above ground but below the footprint’s highest point (*usable basement floor area*), and the part below ground but above the footprint’s lowest point (*unusable basement floor area*). Only the floor area of the part above ground is considered to constitute a part of the target floor area. After the calculations the generated geometry is deleted and the algorithm returns to the building footprint.
5. *Number of floors above basement* is calculated by subtracting the usable basement floor area from the target floor area and dividing the remainder with the single floor area. The result is rounded upwards to the nearest integer. If the usable basement floor area is greater than the target floor area the number of floors above the basement level is nevertheless set to 1
6. *Building geometry* is generated by flattening the building footprint to its lowest point, and extruding the resulting geometry upwards by amount equal to the sum of the footprint’s height before flattening (the difference between the footprint’s highest and lowest point) and the height of the building above the basement as determined by the number of floors above basement and the predefined floor-to-floor height.
7. *Building floor area* is calculated by dividing the queried geometric volume of the building with the predefined floor-to-floor height and subtracting the unusable basement floor area from the result.

8. *Equivalence of the target floor area and the building floor area* is tested.
    a. If building floor area is less than target floor area the algorithm reverts back to the original footprint, adds one floor to the number of floors above basement and goes to step 6.
    b. If building floor area is more than target floor area the algorithm reverts back to the original footprint, reduces the footprint's depth by factor equal to the excess floor area, and goes to step 6.
    c. If building floor area is more than target floor area and building depth is already at its predefined minimum the algorithm reverts back to the original footprint, reduces the footprint's length by factor equal to the excess floor area, and goes back to step 6.
    d. If building floor area is within tolerance of the target floor area the algorithm stops.

5.3.4 General notes on the implementation

In the produced script file buildings under 80 m^2 are not generated (step 1). The building depth values used are interpolated from 8 meters (minimum) to 25 meters (maximum), for target floor areas 100 m^2 to 2500 m^2, and the minimum building depth is given for buildings under 100 m^2 and the maximum for buildings over 2500 m^2 (step 2).

Initially the length of the building is the maximum the building siting and the lot allows, so that for example a building sited to the front of a lot with 4 meter setbacks from the neighboring lots' borders occupies the whole street front minus 4 meters from both neighboring lots (step 3).

The number of floors arrived at by dividing the target floor area with the single floor area is rounded upwards as for reasons seen later it is easier to adjust the building depth and length downwards than upwards (step 5).

The building floor area is calculated based on the building volume and the predefined floor-to-floor height instead of multiplying the single floor area with the number of floors so as to allow for the generation of buildings where the single floor area might differ from floor to floor.

As the extrusion function used allows the generation of tapering geometry the upper floors' area might be less than the calculated single floor area (i.e. the footprint area). This can lead to situations where the generated geometry's total floor area is less than the target floor area, even if the rounding in step 5 is done upwards[17]. This is the situation referred to in step 8a.

The approach to reduce the building depth by factor equivalent to the excess floor area (step 8b) ensures the minimum number of iterations is needed to arrive at the defined target value. Apart from the 8 meter building depth minimum for buildings less than or equal to 100 m^2 the building depth reductions are limited to a global building depth minimum of 10 meters (step 8c).

[17] For example if the target floor area is 1000 m^2 and the footprint area is 334 m^2 the initial number of floors the algorithm tries is 3 (1000 / 334 ≈ 2.99, rounded upwards 3). If due to the geometry the generated third floor however is only 272 m^2 the total floor area is only 940 m^2, 6% less than the target and thus not within tolerance.

If even after the building depth is reduced to the 10 meter minimum there is still more floor space than the target value indicates the length of the building is reduced. This reduction is also done by factor equivalent to the excess floor space (step 8c).

The tolerance value used in the produced script file is 5% (step 8d). As the model is formed iteratively there is a trade-off between the tolerance value and the number of iterations needed.

As the algorithm allows freeform building footprints and volumes, is acceptably accurate also in situations where the building is sited on a steep slope (meaning that a substantial portion of the total floor area might be in the basement), and aims to cover all building typologies from single family houses to skyscrapers, it is somewhat complex.

5.3.5 Notes on the CGA implementation

Implementing the algorithm on CGA language requires the underlying data structure and the CGA operations library to be taken into account. Due to these constraints the algorithm needs to be translated to a linear sequence of simple geometric operations, rather than having the building as an object with attributes for depth, length, and number of floors. The shape tree created by the algorithm therefore does not exhibit any branching.

The lot and the building footprint area (step 1 and 3) refer to the area of their orthogonal top projection rather than their 3D-face area. The CGA shape operations library however does not provide tools to query the projected areas of geometries. To overcome this, the geometries are first flattened after which their 3D-face area is queried. The flattened 3D-face area is equivalent to the orthogonal top projection area. As the original (pre-flattening) geometry is however needed in subsequent steps of the algorithm, the geometry needs to be restored after the operation. CGA's data structure however does not allow the algorithm to return to its previous state. The produced algorithm then needs to return to a state identical to its previous state through geometric operations. If the lot or building footprint geometry would be flattened completely, so that all the vertices would share the same elevation value, it would be very difficult and probably impossible to revert back to the original footprint. Therefore the flattening is done by reducing the height of the geometry to one hundredth of the original. Reverting back to the original geometry can then be done by multiplying the geometry's height by a factor of one hundred. The method is not exact but the margin of error is acceptably low.

Another situation where the algorithm needs to revert back to the original footprint geometry is when it queries a building geometry and finds out it is not within tolerance of the target floor area (step 8a, b, and c). It then deletes the 3D-faces representing the walls and the roof of the building to be left only with the bottom face representing the building footprint. After the bottom face's normal is reversed so that it points upwards the algorithm is in a state identical to where the building geometry was initially extruded from.

The geometry creating operations of the CGA library are mostly aimed at creating 3D-volumes from 2D-polygons. Extending an existing 2D-polygon while keeping the original topology intact can only be done through scaling operations, which are difficult to control in a precise manner. The CGA library however features a number of operations for the precise subdivision of 2D-shapes, due to CGA language incorporating many of the "split grammar" concepts presented by Wonka, et al. (2003). For this reason the approach assumed is to arrive

at the target floor area from above, by gradually reducing the size of the building footprint (steps 8b and c). Each of the building siting configurations requires a separate set of geometric operations to handle the building depth and length reductions.

5.4 Efficiency of the system

The measurement of efficiency of modeling with produced system was conducted in relative terms, comparing the use of the script file in CityEngine 2012 to using ArchiCAD 16. ArchiCAD is a building information modeling (BIM) program mainly used in the architectural design of buildings, but also suitable for other types of modeling tasks. For the author it is the most familiar modeling application and thus gives the fairest comparison to CityEngine in terms of efficiency. The benchmark area to be modeled was delimited to three urban blocks in central Helsinki, comprising of various building typologies. Bird's eye view of the area is presented in Figure 15.

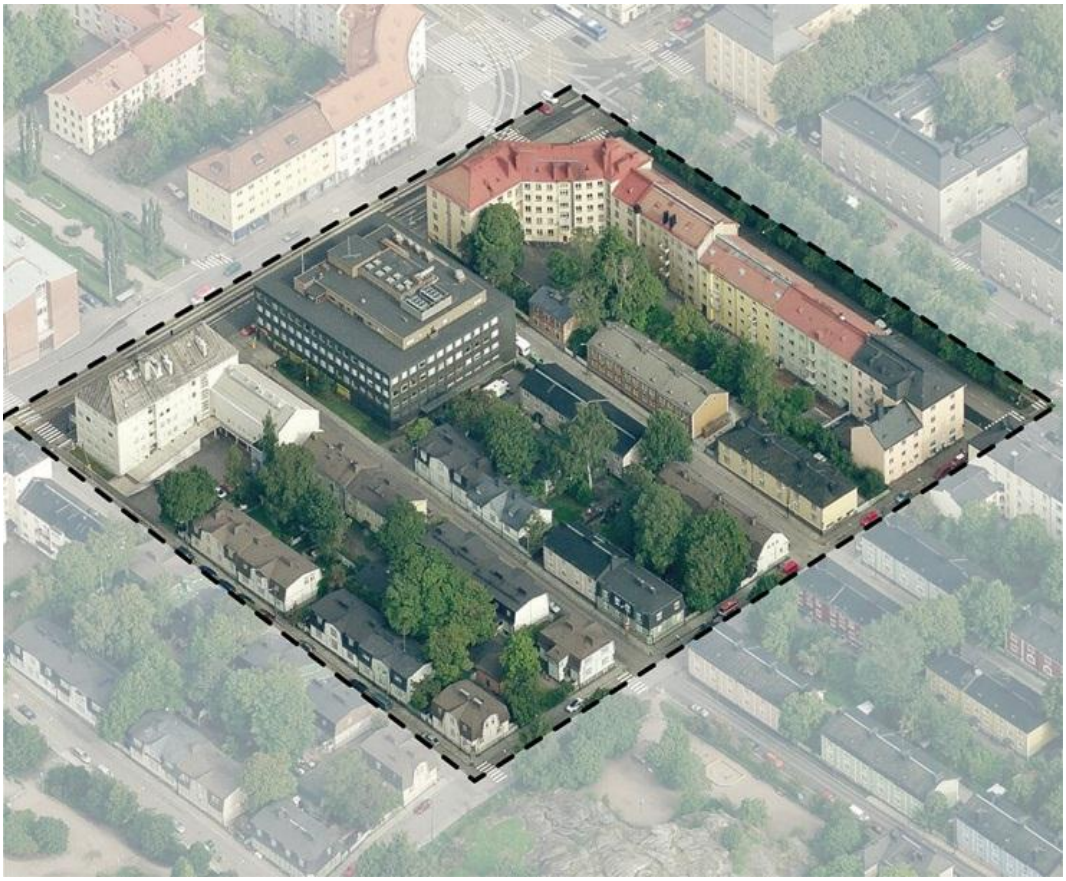

Figure 15 Birds eye view of the benchmark area (adapted from Bing maps)

As argued earlier the approach assumed was to model an existing area in order to eliminate design work from the comparison and focus solely on the efficiency of the tool. Although procedural modeling can be used to model existing areas given feasible input data (for example building footprints with attribute data), the goal of the analysis however is indeed to study the efficiency of the tool applied to modeling work when designing and planning new urban areas. The benchmark area then only works as a visual reference for the modeling work, and represents a "design idea" rather than "input data". Apart from the bird's eye view, a map presented in Figure 16 was imported to both software applications to help in the modeling work.

Figure 16 Map of the benchmark area (adapted from http://kartta.hel.fi/)

As also noted, one of the fundamental aspects of procedural modeling is the limited control over the end result. Modeling an area based on a fixed design idea then is bound to be a balancing act between sufficient fidelity to the design idea and the amount of work incurred. However, contrary to manual modeling frameworks, such as ArchiCAD, even excessive amount of work might not be enough to meet the wanted level of fidelity. The efficiency of procedural modeling is then easily lost by too strict adherence to design ideas.

Despite the two approaches – manual and procedural – being largely incommensurable, in real life projects involving modeling of urban areas the choice would have to be made between the two. The following measurements give some insight into the relative merits of the two in terms of their efficiency and with the aforementioned reservations give a partial answer to the research question presented in the thesis. To simplify the comparisons, no texturing was used in the modeling of the benchmark area.

5.4.1 Baseline measurement

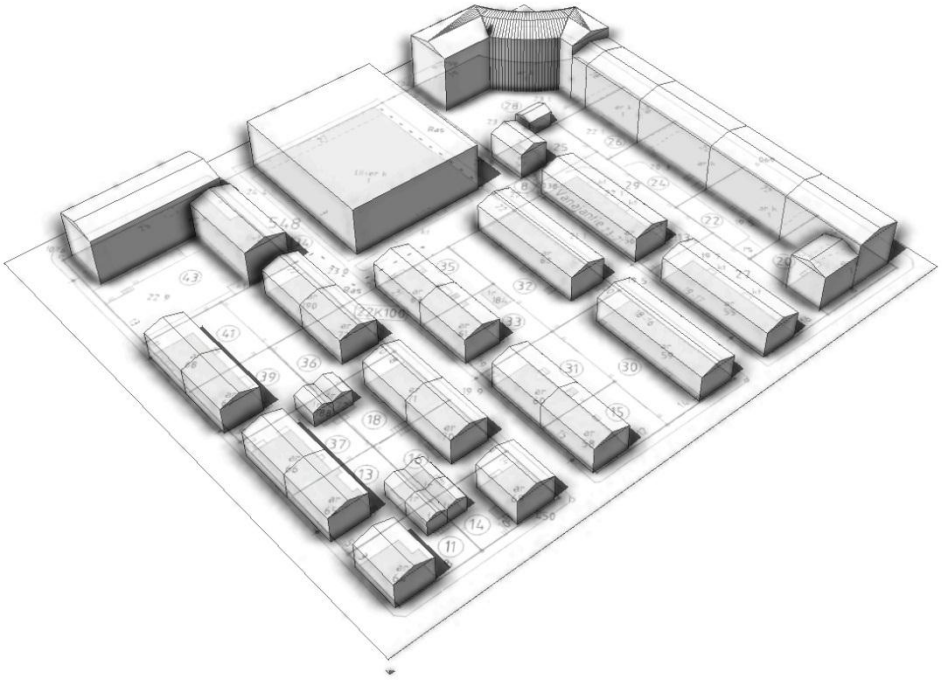

Figure 17 Benchmark area modeled in ArchiCAD

The baseline measurement was done in ArchiCAD 16, using the most efficient workflows known to the author. The buildings, except for the big square shaped office building in the middle block, were modeled using the wall tool with buildings of different heights predefined as wall profiles. The wall tool extrudes geometry of given width and height or section profile between the defined start and end points of the wall. Despite the name of the tool it can then be used to create geometries representing objects other than walls. The office building was modeled as a simple slab object. The gable roofs of the buildings were also modeled using the wall tool with a predefined profile for the roof shape. The basic workflow then consisted of choosing the correct wall profile (for example 3-floor building), defining the start and end points of the extrusion along the side of the building in the reference image, giving the extrusion proper width based on the depth of the building, and creating the roof in a similar manner. As many of the buildings in the benchmark area are more or less identical to each other the above mentioned process was not always repeated but the created geometry was rather copied and pasted to new locations. The resulting model is presented in Figure 17.

5.4.2 CityEngine measurement, 1st iteration

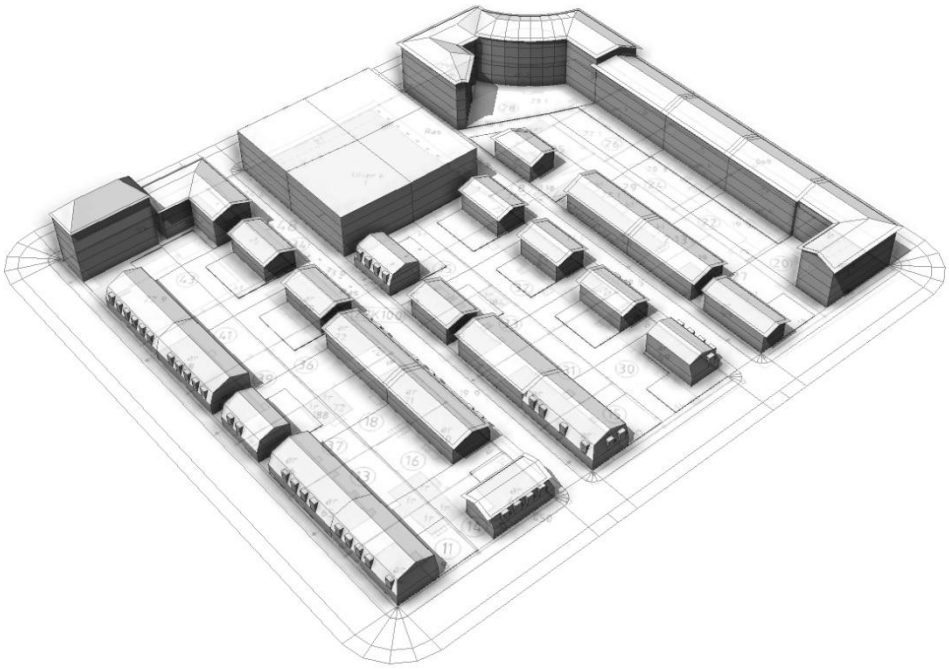

Figure 18 Benchmark area modeled in CityEngine, 1st iteration

The second measurement was done using the produced script file in CityEngine. The general workflow in CityEngine, also used in this modeling experiment, is to draw the street network, give the street segments the desired width, adjust the block subdivision algorithm's parameters to produce suitable lots, assign CGA scripts to the lot and street segment shapes, execute the CGA scripts, and finally to adjust the parameters of the CGA scripts until the produced geometry is as desired. In practice the workflow moves back and forth these steps. The aimed level of fidelity to the benchmark area in the second measurement was set moderately high, such that the area should be easily recognizable from the model but with some deviation accepted. Both the indicator based and parameter based LOCs were used to produce the model presented in Figure 18.

5.4.3 CityEngine measurement, 2nd iteration

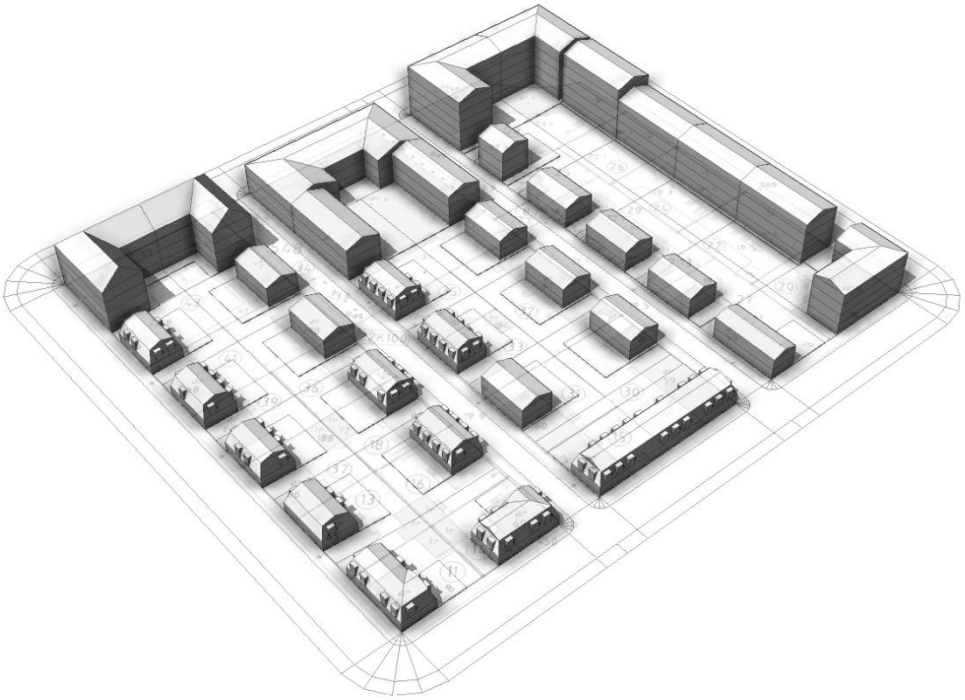

Figure 19 Benchmark area modeled in CityEngine, 2nd iteration

The third experiment was also conducted using the produced script file. Compared to the second measurement the desired level of fidelity to the benchmark area was set lower, mainly aiming to reflect the overall building volume of the area with more or less characteristic building masses. Only the indicator based LOC was used for the third measurement. The produced model is presented in Figure 19.

5.4.4 Number of user interactions

The number of user interactions required for the experiments is presented in Figure 20. As mentioned in Chapter 4 the user interactions were counted with simple software that keeps a log of user activity. The detailed results for each of the experiments were:

- ArchiCAD:
  - left mouse clicks: 380
  - middle mouse clicks: 1
  - right mouse clicks: 0
  - keystrokes: 3609 (of which 3393 were right shift)
- CityEngine 1:
  - left mouse clicks: 401
  - middle mouse clicks: 27
  - right mouse clicks: 4
  - keystrokes: 1774 (of which 1657 were right shift)
- CityEngine 2:
  - left mouse clicks: 230
  - middle mouse clicks: 10
  - right mouse clicks: 4
  - keystrokes: 1090 (of which 1001 were right shift)

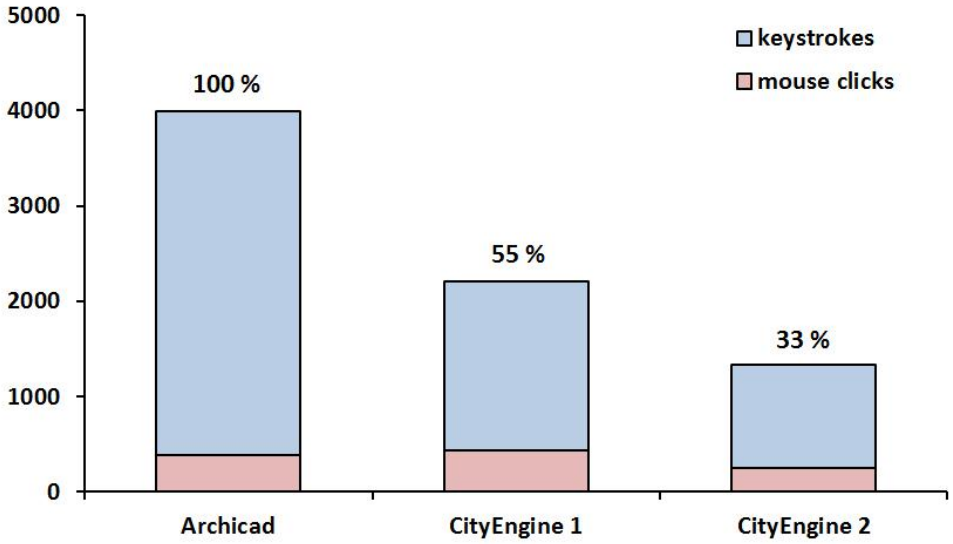

Figure 20 Number of user interactions needed for modeling the benchmark area

6 Discussion of results

Before analyzing whether or not the produced system fills its role, it is worthwhile to consider if the problems it is meant to solve are relevant. The requirements presented in chapter 5 roughly outline these problems stating that there is a need for a planning tool which is more accurate, efficient, and easy, being able to produce higher quality visualizations, and with better analytical capabilities than the existing tools. Importantly all these objectives are intended to be met within a single integrated software framework.

After the relevance of the stated problems has been explored, the ability of the produced CGA script file and the system it is a part of to meet the requirements is analyzed. Due to the choice made in the thesis the evaluation is most detailed in the analysis of the system's efficiency. Other dimensions are analyzed on a more general level. To understand how efforts to improve the system should be allocated, the results of the analysis are prorated either to the CityEngine framework or to the produced CGA script itself. The following is a requirement-by-requirement breakdown of the results of the analysis.

6.1 Accuracy

Accuracy in the context of an urban planning tool can be understood in a number of ways. The ways in which the term seems to have been used by the planners interviewed by Puustinen (2004) relate either to the quality of the background material on which the planning and design is conducted, or to the accuracy of the calculations derived from the produced plans. The term can also be understood to describe the representation of the planning idea, for example on a continuum of a symbol to a zone to a building (though this type of accuracy might better be described as precision). The relevance of accuracy generally depends on the scale of the planning task, such that the larger the area to be planned the less accuracy is needed. Using a single tool then for planning on multiple scales is bound to be under accurate for certain scales and over accurate for others. Conversely it can be argued that a single tool is best suited for planning on a specific scale, the scale being native to the tool. The study by Puustinen then seems to confirm the general demand for accuracy, but the level of accuracy reached in the finished production tool finally dictates the scale where the tool is the most relevant.

Compared to zoning maps (Fig. 21) procedural modeling of urban areas by its very nature represents a higher level of precision in the representation of plans. The precision of the representation then puts pressure on and enables the accuracy of the plan itself to rise as well. For example a zone in a general plan might be given a FAR value to define the permissible building volumes. As the zonal representation is very imprecise (except perhaps for its boundaries) the implications of the FAR value might not be thoroughly considered. Representing the same building volume over the same area with physical buildings might reveal the FAR value to be excessive and put pressure on reducing the value. The building level representation then enables the planner to explore planning alternatives to arrive at a more suitable FAR value, whether or not the final plan representation is kept at the original zone based or replaced with a more detailed one.

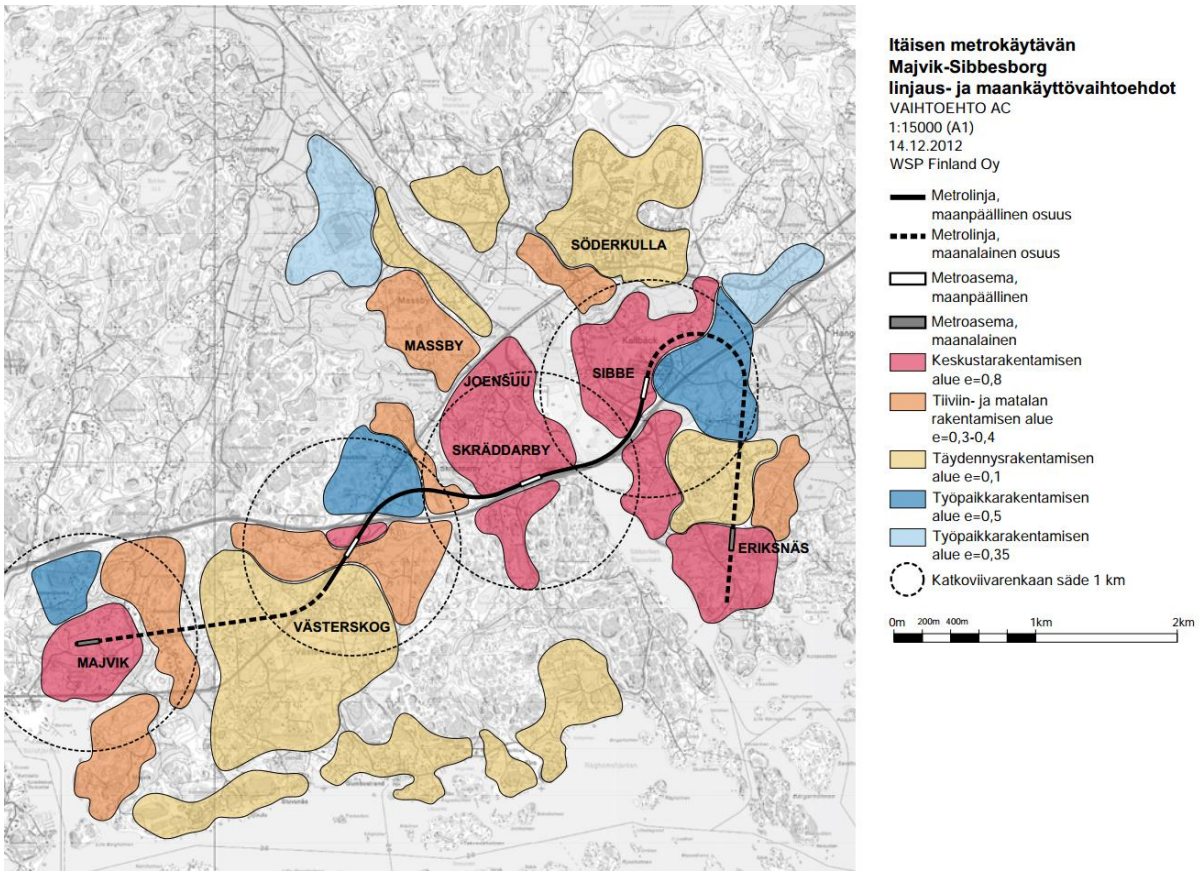

Figure 21 Preliminary zoning map for Sibbesborg (WSP Finland, 2013)

Due to the lack of control in the procedural approach in comparison with manual modeling the accuracy of the production system is significantly less than the accuracy of established CAD and BIM tools. As the tool is not purely procedural, but in the normal workflow rather augments manual modeling with procedural elements, the loss of relative accuracy (in relation to some preconceived design idea) is the most evident in the elements created algorithmically. For example the street centerlines can be placed manually to their desired position, but the lot borders generated by the block subdivision algorithm cannot.

How accurately the buildings generated on LOC0 with the FAR-to-building algorithm conform to what could be considered a realistic building also needs to be considered. The interpolation function used to arrive at the initial building depth and the algorithm in general in most cases seems to produce buildings of realistic proportions. As noted the proportions are kept in balance by selectively adjusting either the height, length, or the depth of the building. In certain circumstances however the selection does not work and the algorithm generates unrealistically shallow building masses. In the current version of the script file the calibration of the initial depth interpolation function is based on intuition. For more realistic buildings the calibration of the function and perhaps of other parts of the algorithm as well could be based on statistical analysis of existing buildings.

To allow for sufficient design freedom in the morphology of the generated blocks, the script file allows the generated buildings to be sited in a number of ways on their respective lots. As the building siting is implemented using a type of an offset function, the array of possible configurations however is limited such that the buildings' walls are always parallel to the lot

borders. In urban design tasks this is clearly a limitation. Even if the siting of the building footprint within the lot area is then somewhat limited, converting the 2D-footprint to a 3D-building however is adequately controllable for most purposes. The extrusion function used allows the generation of tapering geometry such that the floor level from which the tapering starts and its angle can be freely defined. Moreover these definitions can be done individually for each wall of the building. Apart from the general shape of the building generated with the extrusion function, the roof shape can be selected from any of the built-in roof functions offered by CGA. Depending on the type of the roof the functions take parameters such as roof angle, overhang distance for the eaves, and ridge direction. The geometric controllability of the production tool enables the representation of variety of building typologies, and thus reinforces the general applicability of the tool.

Another important dimension of accuracy is the correspondence between the generated geometry and the data reported during the generation process. A number of measures have been taken to ensure the reported data is correct. The data is either related to the size of the lot or the building, and the derivatives thereof. The lot area is calculated as the area of the orthogonal top projection of the lot rather than the area of the lot polygon itself. In steep topography the two might differ significantly. The measures taken to ensure the reported building size is correct relate mostly to the way the basement area is calculated by disregarding the part of the basement below ground. This is roughly in accordance with the Finnish standard for measuring permissible building floor areas.

Many of the shortcomings in the accuracy of the production system are attributable to the weakness of the basic modeling tools in CityEngine. This is evident for example in the lack of a simple measuring tool to analyze distances, areas, and angles. Also the feature common in most CAD systems to input the length of the drawn features numerically on the keyboard is missing.

6.2 Efficiency

The relevance of efficiency in urban 3D-modeling can be thought to be formed through two related but distinct processes. Firstly the efficiency of tools allows cutting down labor costs incurred by extensive modeling projects. Secondly the growth in efficiency allows the adoption of 3D-modeling workflows in projects where due to the expected costs this has not been possible before. Though it can be argued that the budgets and therefore the potential savings in absolute terms are far greater in entertainment industry than in urban planning industry, the scale of the problem can be appreciated by the fact that it took 15 man years to complete the urban models in the movie *Superman Returns* (Müller, et al., 2006).

The measurements obtained in this thesis are an attempt to give numerical indications as to how efficient procedural modeling is compared to more established modeling workflows. Taking the measurements as is seems to indicate procedural modeling of urban areas to be up to three times more efficient than the most efficient workflows in standard BIM software. Applied directly to the production budget of the movie *Superman Returns* the technology's efficiency improvement would´ve then saved 10 man years. The validity of the results of the measurement outside of the setting where they were obtained however is somewhat questionable. The primary complication in comparing the relative efficiencies of procedural and manual modeling workflows is that even though the input components of the calculations (user interactions, in the case of this thesis) are largely commensurable the output components are intrinsically different. Also the equivalence of user interactions can be questioned; is the workload implied by each of the keystrokes or mouse clicks equal? The choice of the benchmark area also has an effect on the result, as some morphological configurations of urban areas are easier to capture with the produced tool than others. The chosen benchmark area with its conventional block structure is generally favorable to CityEngine and the produced script file. Importantly, the conducted measurement concerns only the geometry of the model. For more comprehensive assessment other aspects should also be taken into account, for example the fact that the CityEngine model also provides real time information on the plans floor area.

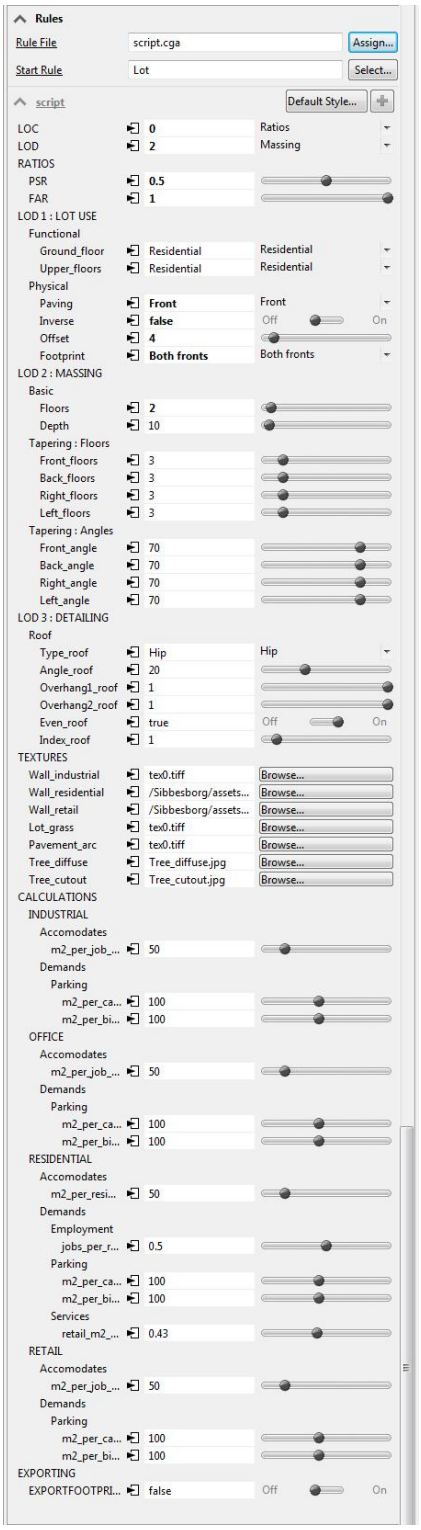

Figure 22 The script file's GUI

As a first approximation the measurement however is valuable, and can be used as a baseline for further improvements to the produced CGA script file. In the current version of the script file the LOC structure is only implemented as a possibility to input the FAR value instead of building height and depth. In essence the efficiency improvement the LOC0-level then offers is rather small as it only eliminates one parameter from a long list of parameters (Fig. 22). Not all parameters are equal though as some are modified much more frequently than others. One way to attain efficiency improvements then is to introduce structures to the file which bundle existing parameters to a single controllable value, much to the same effect as the FAR value controls the building mass parameters. The controllable value might be called for example "style" or "year of construction" and dictate parameters such as the tapering of building mass, roof type and angle, and the type of texturing used.

The measurements and the visual comparison of the produced models also hint at the relative strength of procedural modeling: a lot of detail can be easily added to the model, which if modeled manually would probably alter the relative efficiency between procedural and manual modeling much to the formers favor. In the produced script file the fine details are mostly limited to the randomly placed dormer windows, but could rather easily incorporate other elements as well.

6.3 Ease of use

The ease of use of the produced system directly affects the likely adoption rate of the technology in a given organizational framework, and the time it takes to master the technology once it's adopted. The merits of the system might then go to waste if it is perceived to be too difficult to be put to use.

The underlying control structure and the user interface governing how the user interacts with it are the main elements affecting the ease of use of a system. The degree to which the user experience of the overall production system can be influenced through the CGA scripting language is rather limited, and mostly concerns the definition of the controllable parameters and how they are portrayed in the inspector window of CityEngine's user interface (Fig. 9). The number of controllable parameters is an analogous problem to the one presented earlier about the expressiveness and efficiency of the procedural modeling framework in general. The more expressive the CGA script is intended to be, the more controllable parameters are needed. Due to the limited ways in which the CGA script's controllable parameters can be organized in the CityEngine inspector window, the addition of expressiveness to the script file rather directly leads to a more cluttered user interface undermining the usability of the system. In the produced script file the organization of the controllable parameters in the inspector window is based on simply grouping related parameters together, based on the LOD-module they belong to or categories external to the LOD structure. The parameter groups however are only separated with indentations, and cannot be collapsed. This leaves the whole list of parameters visible at all times making the interface visually clumsy (Fig. 22). The term "controllable parameter" however in the case of a CGA script is quite relative, as even the attributes not visible in the inspector window can be freely modified in the CGA script file itself[18]. Therefore one way to minimize the inspector window clutter would be to hard code certain rarely used parameters into the script file, in practice by writing @Hidden before the declaration of the attribute in the CGA file.

6.4 Visual qualities

As previously argued, the aesthetic qualities of environments can only be credibly analyzed through surrogates such as 3D-models if the visual quality of the surrogate has a high level of realism (Daniel & Meitner, 2001). The qualities to be communicated in the case of an urban 3D-model however are not limited to the purely aesthetical. Every communication need then is best accommodated by a specialized visualization mode. In the produced script file the visualization modes are defined by the LOD-structure, with the LOD1 (Massing) and LOD2 (Detailing) being the most relevant for visualization purposes. The LOD2 corresponds to the level of highest realism, whereas LOD1 deals with mass models which are perhaps best suited for general planning questions, such as the distribution of building volume over the planning area. Depending on the scale of the planning task the static nature of the model might create

[18] Unless the CGA script file is saved as a .rpk binary package as possible from CityEngine 2013 on

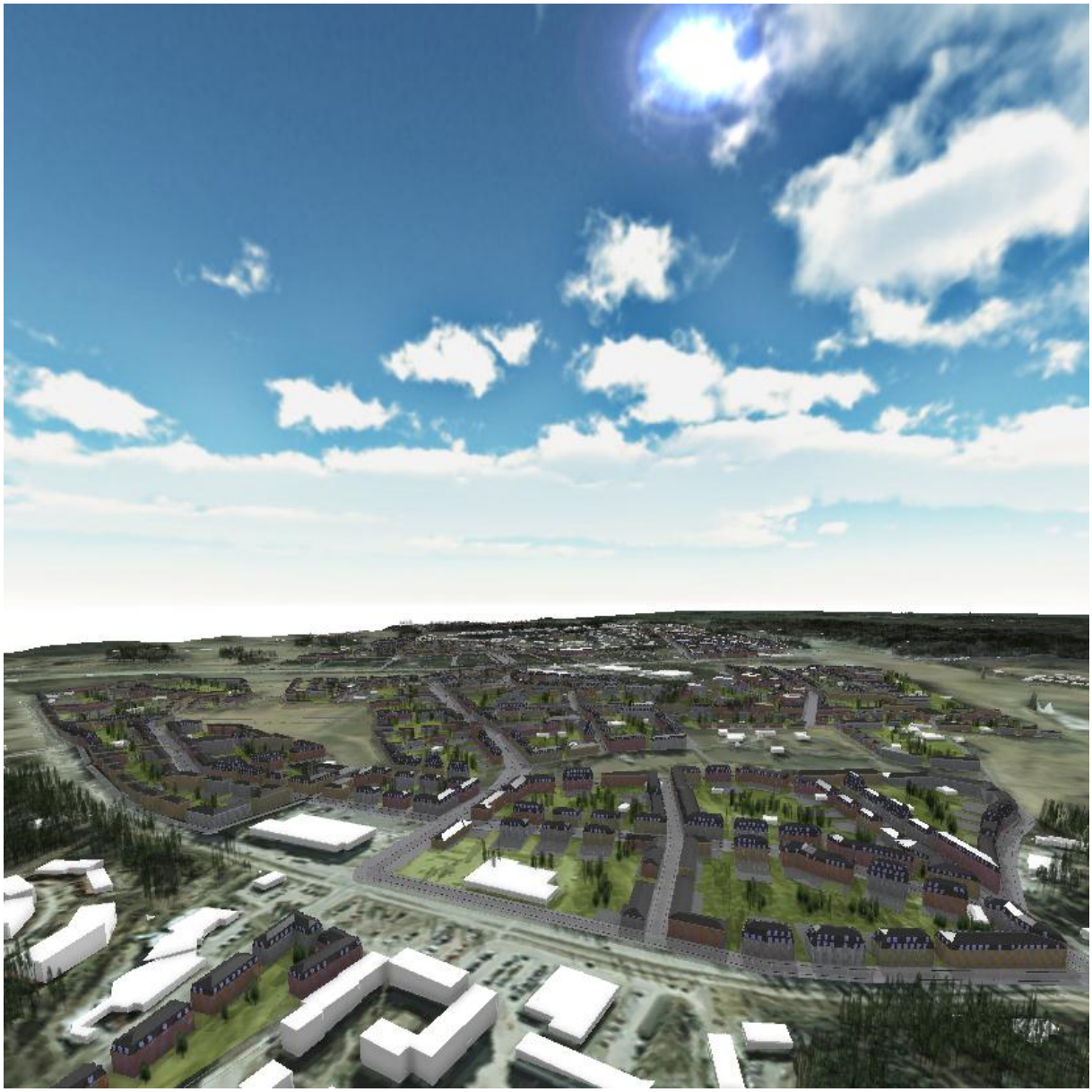

Figure 23 Screenshot from CE showing a part of the Sibbesborg model

false impressions of the nature of the plan. In general level master planning the amount of uncertainty in many cases does not warrant the representation of the plan as physical structures (Salmi, 2006). On the other hand, the consequences of the plans are much easier to understand from such physical representations. Rather than discarding the building level visualization of large scale plans all together it might be worthwhile to consider the possibility of introducing elements of randomness to the visualization.

The level of reached realism on LOD2 in the produced script file has been achieved by applying façade textures on the 3D models of the generated buildings. The realism of the 3D geometries themselves is a combination of feasible building footprints, extrusion function allowing the generation of mansard roofs and other freeform masses, various types of roofs in a variety of angles and overhang distances, and randomly placed dormer windows. The landscaping of the lots includes randomly placed trees, the number of which is determined by

the size of the lot. The visual quality of the model is decent, but clearly there is a lot of room for improvement (Fig. 23). The façade textures could be improved, or the windows and doors could rather be implemented as geometric features. This could be done for example through the geometric operations in CGA language, or with pre-modeled assets attached to the generated model with the CGA insert operation. Similarly secondary building features such as chimneys, balconies, fences, sheds, and other such details might be added to make the model more immersive.

The most promising possibilities in the procedural production system to develop visualization mechanisms to show the uncertainty of the building level representations of plans are related to the stochastic production rules. In essence the stochastic rules allow the introduction of randomized elements in the generation process. The outcomes of the random processes however are not truly random but determined by the random seed value of the initial shape. By updating the random seed[19] the geometry is regenerated with new outcomes. In the produced script file the randomization is still largely unrealized. The only elements randomized are façade texturing (which is randomly chosen from a number of different bitmaps), and the placement of dormer windows (which are generated with a 0.8 probability in each of the possible locations). Clearly, neither is the type of randomization which could portray the vast array of possible outcomes of a general level master plan. To have the intended effect the concept of randomized elements ought to be studied further and implemented to more fundamental aspects of the urban 3D-model. There are however at least two further problems to the approach: How is the random seed updated, and should the visualization be concerned with the goal of the plan rather than the possible outcomes? The first question concerns the medium through which the multitudes of modeling outcomes are communicated. In live situations such as planning charrettes the planner might operate the procedural production system and manually update the random seed to visualize the possibilities contained in the plan. In static visualizations however only one of the outcomes is visible. The second question is more fundamental: is the idea to try to communicate the different possible outcomes of the plan valid in the first place, or should only the outcome deemed most desirable by the planners be visualized?

6.5 Analytical capabilities

In land use planning the demand for analysis to a large extent is derived from the legal obligations to study the plans impact on various factors, such as living conditions and environment, city- and landscape, built environment, and on health and security (Maankäyttö- ja rakennuslaki, 132/1999). The root motivation to analyze plans is to improve and ensure their quality, and thus would exist – to a lesser extent – even without the legal requirements. The amount of analysis conducted depends primarily on the cost of the analysis and its perceived value. The cost is mostly determined by the amount of work required to conduct the analysis, whereas the perceived value of the analysis dictates the permissible cost. The effect the analysis results have depend on whether they reveal aspects of the plan needing further study and whether there is any time to react to the results. The graveness of the concern revealed through analysis and the required reaction time are related such that the graver the problem the easier it is to find time to correct it. In extreme cases the problems revealed have to be corrected in order for the statutory planning process not to be halted. The finer level optimization of plans however relies on tight feedback loops between the analytical and the

[19] by selecting the initial shape and clicking on the "Update Seed and Generate Models" button in the CityEngine user interface

creative planning processes allowing the planner to see the consequences of planning decisions preferably in real time. Analysis results revealing minor weaknesses in a given plan cannot be taken into account if they come in too late stage of the planning process. Both the cost and time factors then call for the seamless integration of analytical capabilities into the planning system.

The analyses possible with the produced script file and the procedural modeling framework in general are divided to those which can be conducted within the program (tight feedback loop) and those which can only be conducted by exporting data from CityEngine into 3rd party analysis programs (loose feedback loop). Apart from the obvious city- and landscape analysis working with a 3D-model enables, also the shadowing of the plan can be easily studied. The analyses made possible by the produced CGA script file are related to its reporting function. Though elementary, the possibility in the produced system to see the floor area of the plan (and the derivatives thereof such as the number of potential residents) in real time is a feature uncommon in most modeling frameworks. The real time reporting function then works well, and can easily be expanded to show different data as needs arise.

The CityEngine framework itself provides a single purely analytical tool as well. The "Analyze graph" tool can be used to analyze graph networks (i.e. the street networks) for their global and local integration, and in-between centrality, which are concepts loaning from the Space Syntax tradition of urban analysis[20].

[20] https://www.bartlett.ucl.ac.uk/graduate/research/space/space-syntax

7 Conclusions and future work

Procedural modeling has been shown to be a relatively accurate and efficient technology for 3D-modeling in urban planning with high visual quality and potential for tight integration with analysis software. The studied implementation of procedural modeling has been formed by the CityEngine platform and a script file written in the software's CGA scripting language. Both of the components of the system are replaceable. The script file can be modified or rewritten altogether, and the CityEngine SDK[21] enables the CGA based production pipeline to be adopted in other modeling software as well. The results of the thesis can then be thought to give first evaluation of the technology with the perspective of further improvements possible.

The qualities of the produced script file and the system it is a part of seem best suited for preliminary studies of land use potentials, where the aim is to estimate the amount of building volume a relatively extensive land area could accommodate and evaluate the impact of it. In the Finnish planning system the corresponding plan type would be the master plan, except for the most strategic types of master plans where physical building visualizations would seem counterproductive even with their preliminary representative nature somehow made clear. At the other end of the spectrum the domain of application for the discussed production system is limited by its lack of accuracy, rendering the use of it in detailed planning too clumsy. The inaccuracies of the system are largely attributable to the crudeness of the basic modeling tools in CityEngine, and thus the domain of application could expand if the CGA based production system would be integrated to more advanced modeling frameworks through the CityEngine SDK.

The effect the technology at its current level of maturity could have on the identified relevant planning level comes through the introduction of 3D-modeling workflows to plans which traditionally have leaned on zonal representations. The possibility to transform the representation type to a much more precise 3D-building based is a direct consequence of the efficiency of the production system. The measurements conducted in the thesis seem to indicate the studied system to offer up to three fold efficiency improvements as compared to more established manual modeling frameworks. The measurements of the type conducted in the thesis are however highly sensitive to changes in the initial premises (regarding for example the produced model's required fidelity to the original benchmark area) and leave a lot of room for interpretation. It seems relatively safe to assume though that the comparative advantage of procedural modeling in relation to manual modeling grows in proportion with the richness of random detail in the model, and declines with the required fidelity to the preconceived design idea.

The impact 3D-modeling based workflows could have on master planning processes are related to the improvement in accuracy and understandability of the produced plans. In the hierarchical planning system used in Finland the more general master plans guide the process of detailed planning. For the guiding mechanism to be meaningful the disparity between the building volumes proposed in the master plan and those permitted in the detailed plans should be limited. Any tools that better enable taking into account the constraints of the planning area in ways efficient enough to be practical on the master planning scale then minimize this disparity and ensure the building volumes proposed on the master planning scale are better

[21] https://github.com/Esri/esri-cityengine-sdk

carried on to the detailed plans and later possibly to physical reality. As previously argued, the precision of 3D-models as compared to zonal plans make them exactly this kind of tool.

The impact related to the improved understandability of plans is closely linked to public participation processes. The dynamics of public participation and its legal foundations in the planning law are very complex, so although it is safe to assume that procedural modeling and the type of plan representations it makes possible do have an effect, the nature of the effect is difficult to predict. It can be argued though that the technology of procedural modeling at least on the representational level of the plans enables the master planning processes to more closely resemble detailed planning. One possible consequence of this is the growth in the number of conflicts the master plans would cause, as the conflict sensitivity of plans seems to grow hand in hand with their tangibility (Peltonen, et al., 2006).

Finally, the potential of the technology to play a role in the integration of the inherently multidisciplinary planning processes is promising. The position procedural modeling could take wouldn't have to be limited to the generation of design data to be analyzed in external software, but the analysis results could also be fed back to the model as parameters guiding the procedural generation process. This feedback loop, despite still being rather loose, can be tightened. The ideal of seamless workflow integrating design and analysis has been presented before, for example under the term *geodesign* (e.g. Miller, 2012, Flaxman, 2010). The results of the thesis hopefully will help reaching this ideal.

References

Al-Kodmany, K., 2002. Visualization Tools and Methods in Community Planning: From Freehand Sketches to Virtual Reality. *Journal of Planning Literature*, 17(2), pp. 189 - 211.

Batty, M., Chapman, D., Evans, S., Haklay, M., Kueppers, S., Shiode, N., Smith, A., Torrens, P. M., 2000. Visualizing the City: Communicating Urban Design to Planners and Decision-Makers [Online]. Center for Advanced Spatial Analysis, working paper series no. 26. London: CASA/UCL.
Available at: http://www.bartlett.ucl.ac.uk/casa/pdf/paper26.pdf
[Accessed 10 April 2014].

Brooks, R. & Matelski, P. J., 1981. The dynamics of 2-generator subgroups of PSL (2, C). *Riemann surfaces and related topics: Proceedings of the 1978 Stony Brook Conference, Ann. of Math. Stud.*, pp. 65 - 71.

Cliffhanger Visuals, 2010. Valorfrit Commercial [Online].
Available at: http://www.esri.com/software/cityengine/industries/valorfrit-commercial
[Accessed 7 Apri 2014].

Daniel, T. C. & Meitner, M. M., 2001. Representational Validity of Landscape Visualizations: The Effects of Graphical Realism on Perceived Scenic Beauty of Forest Vistas. *Journal of Environmental Psychology*, 21(1), pp. 61-72.

Dylla, K., Frischer, B., Mueller, P., Ulmer, A. & Haegler, S., 2008. Rome Reborn 2.0: A Case Study of Virtual City Reconstruction Using Procedural Modeling Techniques. *Computer Graphics World*, 16(25).

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K. & Worley, S., 2003. *Texturing & Modeling, A Procedural Approach*. Third Edition. San Francisco: Morgan Kaufmann Publishers.

Flaxman, M., 2010. Fundamentals of Geodesign. *Proceedings of Digital Landscape Architecture*.

Gregor, S. & Hevner, A. R., 2013. Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, 37(2), pp. 337 - 355.

Harvard University, n.d.. Collection of Historical Scientific Instruments. [Online]
Available at: http://chsi.harvard.edu/ , click "Waywiser" and search with the inventory number "1999-1-0066"
[Accessed 7 April 2014].

Hendrikx, M., Meijer, S., Van Der Velden, J. & Iosup, A., 2013. Procedural Content Generation for Games: A Survey. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, 9(1), Article No. 1.

Kasanen, E. & Lukka, K., 1993. The Constructive Approach in Management Accounting Research. *Journal of Management Accounting Research*, 5, pp. 243 - 264.

Lechner, T., Ren, P., Watson, B., Brozefski, C. & Wilenski, U., 2006. Procedural Modeling of Urban Land Use. *ACM SIGGRAPH 2006 Research Posters*, p. 135.

Lindenmayer, A., 1968. Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. *Journal of theoretical biology*, 18(3), pp. 280 - 299.

Maankäyttö- ja rakennuslaki 132/1999.
Available at: http://www.finlex.fi/fi/laki/ajantasa/1999/19990132
[Accessed 20 April 2014].

Mandelbrot, B., 1977. *Fractals: Form, Change and Dimension*. San Francisco: WH Freemann and Company.

March, S. T. & Smith, G. F., 1995. Design and natural science research on information technology. *Decision Support Systems,* 15(4), pp. 251 - 266.

Miller, W. R., 2012. Introducing Geodesign: the Concept. [Online]
Available at: http://www.esri.com/library/whitepapers/pdfs/introducing-geodesign.pdf
[Accessed 23 March 2014].

Musgrave, K., 1989. Fractal Landscapes. [Online]
Available at: http://www.kenmusgrave.com/Content/html/landscapes.html
[Accessed 7 April 2014].

Müller, P., 2001. Design und Implementation einer Preprocessing Pipeline zur Visualisierung prozedural erzeugter Stadtmodelle (Procedural Modeling of Cities), Zurich: Computer Graphics Laboratory, ETH Zurich.

Müller, P., Wonka, P., Haegler, S., Ulmer, A. & Van Gool, L., 2006. Procedural Modeling of Buildings. *ACM Transactions on Graphics (TOG) - Proceedings of ACM SIGGRAPH 2006*, 25(3), pp. 614 - 623.

Navigant Research, 2012. Building Information Modeling Market to Reach $6.5 Billion Worldwide by 2020. [Online]
Available at: http://www.navigantresearch.com/newsroom/building-information-modeling-market-to-reach-6-5-billion-worldwide-by-2020
[Accessed 8 April 2014].

Paris, Y. & Müller, P., 2001. Procedural Modeling of Cities. *SIGGRAPH '01 Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 301 – 308.

Peltonen, L., Hirvonen, J., Manninen, R., Linjama, H. & Savikko, R. 2006. *Maankäytön konfliktit ja niiden ratkaisumahdollisuudet: Suomalaisen nykytilan kartoitus*, Helsinki: Ympäristöministeriö.

Prusinkiewicz, P., Lindenmayer A., 1990. *The Algorithmic Beauty of Plants*. New York: Springer-Verlag.
Electronic version (2004) available at: http://algorithmicbotany.org/papers/abop/abop.pdf [Accessed 20 April 2014].

Puustinen, S., 2004. *Yhdyskuntasuunnittelu ammattina - Suomalaiset kaavoittajat ja 2000-luvun haasteet*. Helsinki: Ympäristöministeriö.

Reeves, W. T., 1983. Particle Systems—A Technique for Modeling a Class of Fuzzy Objects. *ACM Transactions on Graphics (TOG)*, 2(2), pp. 91 - 108.

Salmi, R.-L., 2006. *Yleiskaavan sisältö ja esitystavat*. Helsinki: Ympäristöministeriö.

Smelik, R. M., de Kraker, K. J. & Groenewegen, S. A., 2009. A Survey of Procedural Methods for Terrain Modelling. *Proceedings of the CASA Workshop on 3D Advanced Media In Gaming And Simulation (3AMIGAS)*, pp. 25 - 34.

Smelik, R., Tutenel, T., de Kraker, K. J. & Bidarra, R., 2010. Integrating Procedural Generation and Manual Editing of Virtual Worlds. *PCGames '10 Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. Article No. 2.

Stiny, G. & Gips, J., 1971. Shape Grammars and the Generative Specification of Painting and Sculpture. *IFIP Congress*, pp. 1460 - 1465.

Synthicity, 2013. Visualization and analysis tools for urban planning professionals. [Online] Available at: http://www.slideshare.net/sblank/synthicity-final-2013-berkeley [Accessed 8 April 2014].

Vaishnavi, V. & Kuechler, B., 2013. Design Science Research in Information Systems. Originally published 2004 [Online]
Available at: http://desrist.org/desrist/content/design-science-research-in-information-systems.pdf
[Accessed 25 March 2014].

Vanegas, C. A., Aliaga, D. G., Wonka, P., Müller, P., Waddell, P. & Watson, B., 2010. Modeling the Appearance and Behavior of Urban Spaces. *Computer Graphics Forum*, 29(1), pp. 25 - 42.

Vanegas, C. A., Garcia-Dorado, I., Aliaga, D. G., Benes, B. & Waddell, P., 2012. Inverse Design of Urban Procedural Models. *ACM Transaction on Graphics (TOG)*, 31(6), Article No. 168.

Watson, B., Müller, P., Veryovka, O., Fuller, A., Wonka, P. & Sexton, C., 2008. Procedural urban modeling in practice. *Computer Graphics and Applications*, IEEE, 28(3), pp. 18 - 26.

Wolski, J., 2010. *Genetic Urbanism - Evolutionary Methods in Urban Design*, M.Arch. Thesis, Aalto Yliopisto.

Wonka, P., Wimmer, M., Sillion, F. & Ribarsky, W., 2003. Instant Architecture. *ACM Transactions on Graphics (TOG)*, 22(3), pp. 669 - 677.

WSP Finland, 2011. Nourish! [Online]
Available at:
http://www.sipoo.fi/easydata/customers/sipoo/sibbesborg/kilpailu/fi/kilpailuehdotukset/76-nourish_.html
[Accessed 7 April 2014].

WSP Finland, 2013. Itäisen metrokäytävän esiselvitys. [Online]
Available at:
http://www.sipoo.fi/easydata/customers/sipoo/files/sibbesborg/selvitykset/sima_raportti_lowres.pdf
[Accessed 8 April 2014].